NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

AN ANALYSIS OF TWO LAYERS OF ENCRYPTION TO PROTECT NETWORK TRAFFIC

by

Ryan Ware

June 2010

Thesis Advisor:	George Dinolt
Second Reader:	Jennifer Guild

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2010	Master's Thesis

4. TITLE AND SUBTITLE: An Analysis of Two Layers of Encryption to Protect Network Traffic

5. FUNDING NUMBERS

6. AUTHOR(S): Ryan T. Ware

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)
In this thesis, we attempt to analyze the effectiveness of defense-in-depth mechanisms. As an example of defense-in-depth, we study two layers of encryption to protect network traffic. At a quick glance, two layers of encryption appear to provide some strong security benefits including increased host- and network-level security, increased cryptographic strength, and a backup layer of encryption. However, intuition and quick glances should not be relied upon in the field of Information Assurance. The intent of this thesis is to quantitatively show the increase in security the extra layer of encryption provides and to compare this information with the cost of the extra security. This thesis proposes two architectures with one layer of encryption and and several architectures with two layers of encryption. It quickly compares these architectures and then starts a more in-depth analysis of the best two-layer architecture using Fault Tree Analysis. The thesis presents the results from the study, provides some recommendations based on the results, and discusses future work in this field.

14. SUBJECT TERMS	15. NUMBER OF PAGES
encryption, computer security, network security, architecture, fault tree analysis, defense-in-depth	97
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

AN ANALYSIS OF TWO LAYERS OF ENCRYPTION TO PROTECT NETWORK TRAFFIC

Ryan T. Ware
Civilian, Department of Navy
B.S., New Mexico Tech, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2010

Author:

Ryan Thomas Ware

Approved By:

George Dinolt
Thesis Advisor

Jennifer Guild
Second Reader

Peter J. Denning
Professor and Chair, Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, we attempt to analyze the effectiveness of defense-in-depth mechanisms. As an example of defense-in-depth, we study two layers of encryption to protect network traffic. At a quick glance, two layers of encryption appear to provide some strong security benefits including increased host- and network-level security, increased cryptographic strength, and a backup layer of encryption. However, intuition and quick glances should not be relied upon in the field of Information Assurance. The intent of this thesis is to quantitatively show the increase in security the extra layer of encryption provides and to compare this information with the cost of the extra security. This thesis proposes two architectures with one layer of encryption and and several architectures with two layers of encryption. It quickly compares these architectures and then starts a more in-depth analysis of the best two-layer architecture using Fault Tree Analysis. The thesis presents the results from the study, provides some recommendations based on the results, and discusses future work in this field.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

In this thesis, we attempt to analyze the effectiveness of defense-in-depth mechanisms. As an example of defense-in-depth, we study the use of two layers of encryption to protect network traffic. At a quick glance, two layers of encryption appear to provide some strong security benefits including increased host- and network-level security, increased cryptographic strength, and a backup layer of encryption. However, intuition and quick glances should not be relied upon in the field of Information Assurance. The intent of this thesis is to quantitatively show the increase in security the extra layer of encryption provides and to compare this information with the cost of the extra security. The two layers of encryption is meant to be one example of a defense-in-depth architecture. We use this to help use study and explore the effectiveness and costs of defense-in-depth mechanisms.

Several methods of formal analysis are presented in this thesis. One method is Formal Methods which is probably the method that can provide the most assurance but also requires the most effort. Metrics are another useful method but are difficult to produce. Heuristic methods such as Fault Tree Analysis and Attack Graph Generation are also useful. Each of these methods are briefly described along with their strengths and weaknesses.

This thesis proposes two architectures with one layer of encryption and and several architectures with two layers of encryption. The two one-layer architectures are host-to-host and gateway-to-gateway. The two-layer encryption schemes are host-to-host, gateway-to-gateway, and a hybrid approach. The thesis quickly compares these architectures discussing their strengths and weaknesses. The hybrid approach with one layer of encryption at the host level and one at the gateway level is determined to be the most useful and interesting.

The hybrid encryption architecture is studied more in-depth using Fault Tree Analysis. The primary intent of the analysis is to determine how beneficial the second layer of encryption is as a backup layer of protection if one layer were to fail. To help determine this, a better understanding of the dependencies in the system is necessary. Fault Tree

Analysis is useful for showing dependencies. However, the fault tree provided in this thesis was not able to show any quantitative results. More research must be conducted to provide more results. Further research topics include further developing the fault tree provided in the thesis, useful metrics related to the topic such as probabilities of failure or costs, and describing and understanding dependencies within a computer system. The results of the analysis are not sufficient to show anything quantitatively, but they are useful for providing some skepticism and the need for in-depth analysis regarding defense-in-depth security mechanisms.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. George Dinolt, for his constant support, supply of knowledge, and different view of things. I greatly appreciate his desire to push me and encourage me to struggle with difficult topics. I would also like to thank Jennifer Guild who provided me with such an amazing opportunity and support. Finally, I must express my gratitude for my amazing wife, Karen, who has been incredibly patient and encouraging throughout this entire process.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The primary goal of this thesis is to critically analyze a system that employs defense-in-depth security mechanisms. To do this, we provide a simple example system and a security policy for the system. The policy is very generic and basic: protect data while in transit across unprotected networks from being deduced or modified without proper authorization. The defense-in-depth mechanism examined is the use of two layers of encryption to protect network traffic. In this thesis, we attempt to determine if the second layer of encryption is more secure than the use of just one layer of encryption. Thus, we examine and compare both methods of encryption throughout the thesis.

Common sense makes the use of two layers of encryption, and many other defense-in-depth mechanisms, appear to be significantly more secure than a single layer of protection. The network as a whole should be better protected because of the added difficulty of having to go through two layers of encryption to inject traffic. Host header information should be better protected from some forms of traffic analysis attacks. The cryptographic strength of the protected data should be stronger, and the second layer of encryption acts as a backup for the first layer. If one layer were to fail or become compromised, the second layer should continue protecting the data. However, it is not sufficient to simply let common sense dictate security policies and implementations.

Systems and security mechanisms must be systematically evaluated. This thesis will briefly discuss several methods for critically thinking about and evaluating systems. One of the methods of analysis mentioned will then be applied to the problem of determining the increase in security by adding a second layer of encryption to protect network traffic.

Chapter III of the thesis introduces several ways of analyzing computer security mechanisms. An ideal way would be to use Formal Methods to prove certain properties about the system. Metrics may provide useful numbers for making comparisons between things. Finally, several heuristic methods exist for thinking about a system and gathering

and organizing information about the system in a structured manner for critical analysis.

Each method has its own strengths and weaknesses. If applied properly, Formal Methods can provide the most assurance regarding a certain security feature that may exist in a system. However, this method is very costly. It often requires a great deal of time and demands that the people doing the Formal Methods work understand Formal Methods and the system being evaluated.

Security metrics that provide meaningful numbers are very important for quantitatively showing the strength of a system or one of its subcomponents. Good metrics provide information in a condensed and easily understood manner. This information may make it easier to make comparisons between the security of systems. Unfortunately, good security metrics are difficult to produce for a number of reasons which will be discussed further in later chapters.

Another way to evaluate a security mechanism is to systematically document and study its vulnerabilities and threats. There are a few heuristic methods that are occasionally employed to help. These methods include: Fault Tree Analysis and attack graph generation. These methodologies can be very flexible and rely upon human understanding and input. It can benefit the analysis by allowing for humans to apply their ingenuity more easily throughout the analysis process. However, this also means that the analysis may suffer from a lack of human understanding, insight, and ingenuity.

Every one of these methods suffers, to one degree or another, from the same thing: human error. Even properties that are proved correctly may be based on bad assumptions and may not be accurate or useful (Lowe, 1996). The models upon which the proofs are based are not the system and may not describe the appropriate system properties. The metrics may be incorrect or not useful. We do not always know what numbers may be useful, relevant, or available. The formulas or numbers may be biased or skewed in some fashion. The heuristic methods may too many important architectural features. Inappropriate judgments may be made. The system may not be understood well enough. There may be a disconnect between the personnel working on the analysis and those working on the actual

system.

It is important to understand the strengths and weaknesses of each of these methods to help provide a stronger understanding of how to use these methods. One area of the analysis on one system may benefit more from the application of one method over another or a prudent combination of the methods. This thesis will provide some insight as to when each of these analytical techniques may be useful.

In Chapter IV, we apply Fault Tree Analysis to determine if two layers of encryption provide more security than one layer. The thesis attempts to produce quantitative measurements and information that show this. However, showing an increase in security is not sufficient. We must also ask the following question: is the increased security worth the extra cost incurred by adding the extra security mechanism? In general, this question is difficult to answer. This thesis explores this question briefly.

We conclude by looking again at the benefits that one might hope to attain via the addition of the second layer of encryption. Each of these benefits are accompanied by a brief summary of relevant information provided in earlier chapters that help to determine the increased strength of the system. The chapter quickly reviews the information gained from the Fault Tree Analysis and provides some future work to help procure more useful information. Finally, the chapter ends with some recommendations regarding defense-in-depth architectures and different system security architectures that may be useful the example studied in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter discusses background information relevant to the thesis topic. Multiple encryption is introduced along with some of its benefits and weaknesses. Protocols that help facilitate encryption of network traffic are also presented in this chapter. Finally, basics related to Fault Tree Analysis are provided here.

A. MULTIPLE ENCRYPTION

When applied properly, cryptography provides a certain amount of confidentiality for data. However, the confidentiality provided is limited because of any number of reasons. Sometimes it is because an attacker has discovered the keys for the encryption or has found and exploited a weakness in the cryptographic algorithm or implementation to decrypt data. As such, computer security researchers attempt to carefully protect the key and make the key more difficult to brute force by increasing the key space or increase the strength of the cryptographic algorithm. Some of these attempts to increase the security involve multiple encryption.

1. Double Encryption

One way to improve the security of a block algorithm is to encrypt a block with one key and then encrypt the resulting ciphertext with a different key. Let C be the ciphertext, P be the plaintext, K_1 and K_2 be key one and two respectively, E represent the encryption function, and D represent the decryption function. With this style of encryption, you get the following process:

$$C = E_{K_2}(E_{K_1}(P))$$

$$P = D_{K_1}(D_{K_2}(C))$$

We might expect the resulting ciphertext to be significantly harder to brute force. Instead of taking 2^n attempts to brute force an n-bit key, the double encrypted block should take

2^{2n} attempts (Schneier, 1996).

Merkle and Hellman proved this to be untrue for a known-plaintext attack (Merkle & Hellman, 1981). They used a meet-in-the-middle attack to reduce the number of attempted encryptions from 2^{2n} to 2^{n+1}. This attack requires the analyst to know P_1, P_2, C_1, and C_2 such that

$$C_1 = E_{K_2}(E_{K_1}(P_1))$$

$$C_2 = E_{K_2}(E_{K_1}(P_2))$$

For each K, the attacker encrypts P_1 with E_K and stores the result in a database. After encrypting P_1 with all possible K's, the attacker computes $D_K(C_1)$ for each K and looks for the same result in the database. If there is a match, it is possible that the attacker now has K_1 and K_2. He can use P_2, K_1, and K_2 to check (Schneier, 1996).

Several papers have been published that provide optimizations of Merkle and Hellman's technique that further reduce the amount of work and storage that is needed for the attack to work. While Merkle and Hellman demonstrated their work on DES, their attack can work on any block encryption algorithm (Schneier, 1996).

2. Cascading Multiple Block Algorithms

Cascading multiple block algorithms encrypts one block of plaintext with multiple algorithms using different keys for each encryption. For example, you could encrypt P_1 with algorithm A using K_A and then encrypt the resulting ciphertext with algorithm B and K_B. Combining multiple algorithms like this may introduce some weaknesses or vulnerabilities in the system because of possible interactions that may not be noticed or fully understood. However, this may only be true if the different keys are somehow related to each other. If the keys are independent from each other, the encryption scheme should at least be as strong or as difficult to break as the first algorithm used in the process (Schneier, 1996).

B. IPSEC

IPsec is designed to provide extra protection over a normal IP packet to any protocol or data that may be carried by an IP packet including another IP layer. This extra security includes "access control, connectionless integrity, data origin authentication, detection and rejection of replays (a form of partial sequence integrity), confidentiality (via encryption), and limited traffic flow confidentiality" (Kent & Seo, 2005).

IPsec has two protocols: Authentication Header (AH) and Encapsulating Security Payload (ESP). AH provides only authentication. It does not encrypt network traffic. ESP has similar functionality to AH but can also be used to encrypt data (Kent & Seo, 2005).

Each protocol can be used in two different modes: transport mode and tunnel mode. Transport mode is used to provide protection for the layer encapsulated within the IP layer. Tunnel mode protects tunneled IP headers and above. Tunnel mode encapsulates original IP packets within a new packet that includes an IP header and an IPsec header (Kent & Seo, 2005).

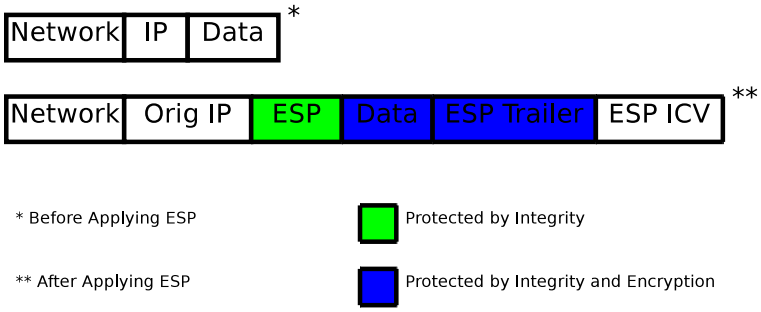

Figure 1: IP packet before and after applying ESP in transport mode.

The protection provided by IPsec may be applied in several different manners. A single IPsec tunnel may be used to protect all IP traffic sent between two security gateways. Each host may have tunnels established for other specified hosts or gateways. Or, specific ports may be protected on a given set of hosts (Kent & Seo, 2005).

Cryptographic keys required by IPsec may be supplied manually or provided by automated means such as with the use of the IKEv2 protocol (C. Kaufman, 2005).

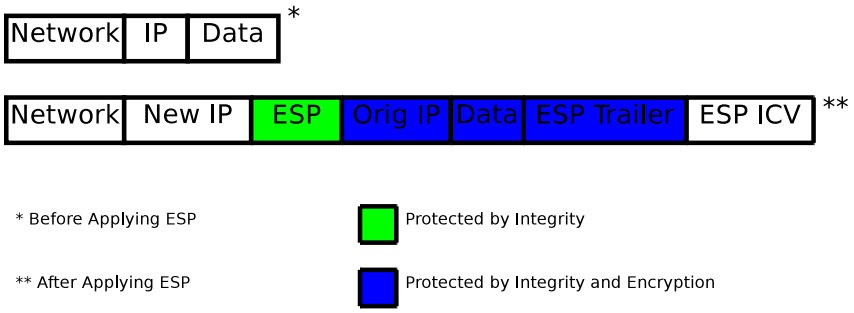

Figure 2: IP packet before and after applying ESP in tunnel mode.

C. OTHER TUNNELING PROTOCOLS

There are several protocols similar to IPsec. Some of the more popular protocols include the Secure Shell (SSH) (Ylonen & Lonvick, 2006) and TLS (Dierks & Allen, 1999). These protocols have slight differences between them. Their primary goals are to provide cryptographic services for network traffic, tunneling of network traffic, and encapsulation. IPsec is designed to sit on top of the IP layer where as the above two protocols are intended to reside above some reliable protocol such as TCP.

There are other protocols that do not provide encryption capabilities but can encapsulate and tunnel network traffic. These protocols include Point-to-Point Tunneling Protocol (PPTP) (Hamzeh et al., 1999), Layer Two Tunneling Protocol (L2TP) (Townsley et al., 1999), and Generic Routing Encapsulation (GRE) (Farinacci, Li, Hanks, Meyer, & Traina, 2000). These protocols tend to be run on lower levels of the network stack such as at layer 2.

D. FAULT TREE ANALYSIS FUNDAMENTALS

A fault tree consists of several symbols that represent events and their relationships to other events in the tree. These events are usually faults but may also be normal or external. They are connected to each other through gates such as the AND- and OR-gate. The top node of the tree is the main fault that is to be analyzed. Child nodes are events that give rise to the main fault.

The events in a tree are represented by one of five different event symbols. The types of events that may be represented are as follows: basic event, conditioning event, undeveloped event, external event, and intermediate event. A basic event is an initiating fault that does not need to be analyzed further. A conditioning event is used to indicate when specific conditions or restrictions must be met. Undeveloped events are not further developed due to a lack of information or because they are of insufficient consequence. Normally occurring or expected events are represented by the external event symbol. An event that arises from one or more events that occur prior to the event are represented by the intermediate event symbol.

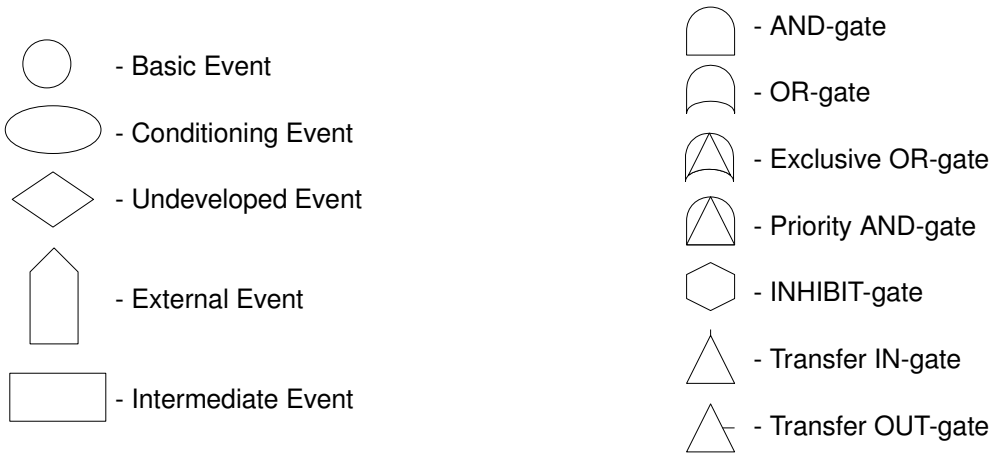

Figure 3: The first column lists the event symbols. The second column illustrates the gate symbols.

The relationships between events are represented by five gate symbols. These gates are the AND-, OR-, Exclusive OR-, Priority AND-, and INHIBIT-gate. An AND-gate is used to show that one or more events must occur before the event represented in the parent node may occur. An event that may manifest itself from the occurrence of one or more preceding events is connected to those events via an OR-gate. The Exclusive OR-gate is used to show an event may occur if only one of the input events occurs. If a specific sequence of events must occur before the output event occurs, a Priority AND-gate may be used along with a conditioning event symbol to the right of the gate. An INHIBIT-gate

is used when the output event takes place if the single input event occurs in the presence of an enabling condition. This enabling condition is represented with a conditioning event symbol to the right of the INHIBIT-gate. Along with these five gate symbols, two transfer symbols, TRANSFER IN and TRANSFER OUT, are used to help organize the tree by indicating that the tree is further developed at the corresponding transfer symbol.

To help with analysis of the system, each event in a fault tree can be associated with its probability of occurrence. Calculating probabilities can be very difficult, and this thesis does not go into great detail on the subject. However, some basic information may prove useful to show how to apply the information. Let us assume event A can only occur if events B and C occur. Given the probabilities of B and C occurring, we know the following:

$$P(A) = P(B) * P(B|A).$$

Let us also assume event B occurs if either D or E occur. The probability of B occurring is as follows:

$$P(B) = P(D) + P(E) - P(DandE).$$

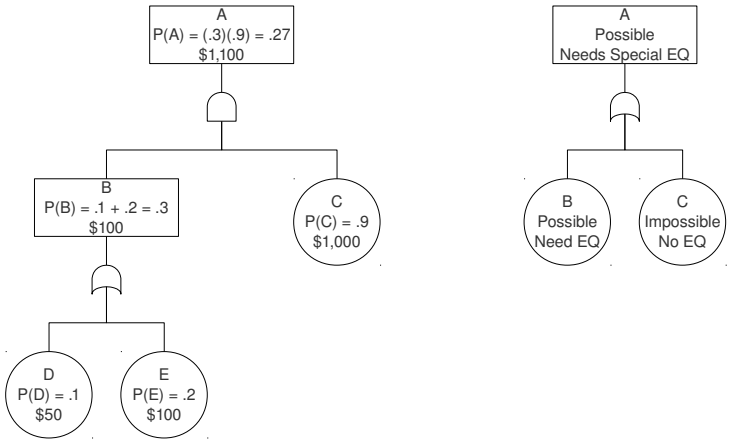

Figure 4: Fault trees providing examples for applying probabilities, costs, and other information.

Other information can be applied to the tree such as the cost of implementing an

attack to provoke a failure event. In the example provided above, the costs of A and B can be determined as follows:

$$Cost(A) = Cost(B) + Cost(C) = \$100 + \$1,000 = \$1,100.$$

$$Cost(B) = min(Cost(D) + Cost(E)) = min(\$50 + \$100) = \$100.$$

The minimum cost of two events connected by an OR-gate is chosen because we often want to focus our efforts on securing against easy attacks.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CRITICAL ANALYSIS

We must remember that "absolute security is no more possible in computer systems than it is in bank vaults" (Denning & Denning, 1979). There are numerous ways in which a computer system may become compromised. The system may have no security or gaping holes in their security. The personnel may be socially engineered to provide unauthorized disclosure of important information. The physical defenses may be compromised leading to the downfall of the computer system. The software itself may fail or become compromised.

This software may fail or have vulnerabilities in it which cause it to be compromised for any number of reasons. Those desiring the software may have not properly communicated their requirements, may have conflicting requirements, or not fully developed the requirements for the system. The software designers may have made the software too complex to fully understand or to test properly. The developers may have used improper coding techniques or may have not understood the requirements properly.

Knowing that absolute security is impossible for "large" systems, we attempt to improve the security of the system by adding in security features or mechanisms. We must strive to ensure these security mechanisms are beneficial and worth the cost of implementing the additional security. To do this, we must attempt to gauge the strength of the security they provide. We must also ensure that the underlying system within which these security features reside do not compromise the security mechanisms.

Several methods may be utilized to help with the analysis. Formal Methods attempt to prove security properties pertaining to a system or protocol. Metrics can be used to provide quantitative measurements that describe the security of the mechanism. Several heuristic methods exist that help to systematically analyze the strengths and weaknesses of the system. We discuss each of these methods including some of their strengths and weaknesses to help determine when and how to use each method.

A. FORMAL METHODS

In "Formal Methods and Their Role in Digital Systems Validation for Airborne Systems," John Rushby described Formal Methods as being the "use of mathematical modeling, calculation, and prediction in the specification, design, analysis, construction, and assurance of computer systems and software" (Rushby, 1995). Formal Methods models one or more aspects of a system and then attempts to deduce or prove the existence of certain properties within the system. This approach differs significantly from attempting to test a system with a set of inputs. One can only test a small fraction of the possible inputs of most systems, making it impossible to prove the absence of vulnerabilities or faults in a system or mechanism. Formal methods, however, can prove certain properties to be true without needing to test every input (Rushby, 1995).

Formal Methods are incredibly flexible and can be applied in a number of different ways. Their application can range from informal proofs done by hand to formal proofs that are run or facilitated by software. One or more components of a mechanism may be evaluated in this manner. A select number of properties may be tested for their existence. Formal Methods may be applied at any stage in the life-cycle process, and the level of abstraction may vary significantly depending on the desired results. Formal Methods may be costly and difficult, so careful thought must be made as to how they are applied.

Rigorous formal proofs will require significantly more time, effort, and expertise than informal proofs done by hand.

The components to which Formal Methods are applied is important. It may be more useful to apply Formal Methods to "the hardest and most difficult problems–where traditional methods are ineffective or unavailable. Examples of hard problems are those involving distributed and concurrent execution and, especially, redundancy management" (Rushby, 1995). If unimportant or simple components are evaluated, significant amounts of resources may be wasted.

It is important to test for the right properties. These properties may include checking if the system can deadlock, survive any single fault, or respond within certain time frames

(Rushby, 1995). Testing for the right properties is like needing to ask the right questions. If the right questions are not asked, the entire evaluation process may not be nearly as effective as it could be.

Formal Methods may be applied at almost any phase in the life-cycle of a system. For example, they may be applied to check the development of the requirements and specifications of the system, the design of the system, or the implementation of the system. If the implementation is tested, errors that may arise due to programming errors may be caught and addressed. However, many problems arise due to misunderstood, inconsistent, underdeveloped, or improperly developed requirements (Rushby, 1995). While these problems will often be caught when checking the implementation, the problems may be significantly easier and cheaper to fix if they are caught earlier. This is another feature that helps to set apart Formal Methods from other test and evaluation methods.

It is extremely difficult to execute and test requirements and specifications. Formal Methods "can help overcome this difficulty by allowing early specifications to be challenged and explored through theorem proving" (Rushby, 1995). So, it may be beneficial to apply this methodology earlier in the life-cycle process to check the requirements of the system.

An appropriate level of abstraction must also be applied to the model for evaluation purposes. Too much abstraction may cause the Formal Methods to miss vital information and skew the results. Too little abstraction may waste valuable resources such as time and personnel. An incorrect abstraction will generate incorrect results that may be difficult to detect even with human review.

There are two important properties one must be aware of when applying abstraction to a system or model: composition and refinement. Composition and decomposition involve the combination or separation of components. Individual components are generally easier to evaluate because they are simpler than their composition. In non-trivial systems, components often affect each other. It is important to not only check individual components for certain properties but to also check their composition for properties. Combining

subsets of a system may reveal new properties of the system or prove that certain properties are no longer held.

A refinement of a model or object possess more detail than the object from which it was refined. Several layers of abstraction may be used throughout the life-cycle of a system. For example, the creation of a system generally starts with a simple goal or small set of objectives. It then progressively becomes more detailed as it transitions to the following: a set of requirements and specifications, the design of the system, and the implementation of the system. The implementation of a system is the refinement of its design. It is important to ensure that every property within one layer of abstraction is present in its refinement. Similarly, properties within the refinement should map back to the more abstract layer. Without checking the refinement of a system, properties may accidentally be removed, modified, or added which may cause problems.

Many problems arise when Formal Methods are applied because there exists a gap between the Formal Methods and the implemented system. There are usually two groups of people working on the "same" thing: the group working on the Formal Methods and the group working on the implementation. Both groups must communicate what is going on with their work and must understand what is happening. A lack of information or understanding may introduce errors in either part of the project. This increases the gap or differences between the model of the system with which the Formal Methods group works and the actual system.

Since Formal Methods deal closely with models and not the actual systems, differences between the model and the system exist. The models may be incomplete because of an inappropriately applied abstraction, or they may just be incorrect abstractions of the proposed system. These differences may result in inaccurate predictions about the system. To help prevent problematic issues from arising, human review and validation is necessary.

The personnel conducting the review as well as those generating the model and running the software must be familiar with the system and have a strong background in Formal Methods. They must have a good understanding of how to properly abstract out

information and apply Formal Methods at the right times and places. Even people with a great deal of experience in the field make mistakes.

Mistakes made in the Formal Methods process may be reduced by having more human review, education, experience, and time. These mistakes may be reduced even further with the use of software that has been designed to help with the proving process. Regardless of the efforts made, mistakes may still be made and results may not be accurate or comprehensive. However, formal methods, if applied well, will still help to find faults in the system.

B. METRICS

The field of information assurance metrics attempts to provide another means of moving computer security from a "black art" ruled by intuition to a field of science with quantifiable measurements. Applying a metric should result in a value that will help compare two or more subjects. This will help make more informed decisions as to which security mechanisms to implement within a system.

The term metric is not always agreed upon as being the useful term to apply to the subject. Instead of the term metric, other similar terms include "measure, score, rating, rank, or assessment result" (MITRE, 2002). This thesis will refer to the value as being a metric since it seems to be the most prominent term used.

1. Example Metrics

While the field of computer security metrics is rather new, there are a number of metrics that are now available and utilized. Some metrics deal with gathering information about an existing system. They may use numbers related to known vulnerabilities for a given application or set of software. Some may use the number of personnel with a certain kind or amount of training. Some metrics may rely upon the number of applications that exist in the system or the number of open ports. Other metrics include the number of incidents that occur in a certain time-frame, mean-time to incident discovery, and the percentage of patch compliance.

Some metrics attempt to measure the cost and risk of a system before it has even been designed and built to help determine future requirements. These metrics may be based on how critical the system and its intended function are perceived to be for the mission or other systems (MITRE, 2002). Examples of these metrics include the categorization scheme of a system as described in FIPS 199 (NIST, 2004) and DoDD 8500.1 (DoDD, 2002). Other metrics along these lines may attempt to determine the scale of the system to explore requirements such as amount of software, encryption keys, users, and hardware.

Other metrics may be determined during the design, development, implementation, and operational phases of a system (MITRE, 2002). These metrics are usually based on the processes followed and requirements met during these phases. If more rigorous design and development methods are used, the system should fair better within a metric. Example metrics are derived from certification levels or simply attaining compliance with known standards such as ISO 9000 (www.iso.org) and SSE-CMM (www.sse-cmm.org). If a system is designed to use certain features known to provide more security, the system may score higher on metrics as in the Common Criteria (CC) and its Evaluation Assurance Levels (EAL) (Criteria, 2009).

Metrics based on testing may also be provided. One indication of code quality is the number of bugs per lines of code. There are several ways to look for errors in source code including peer review, systematic testing of a relatively small set of inputs, and fuzz testing. Fuzz testing generates random or pseudo-random inputs to test the software. These testing methods introduce new metrics such as test coverage of the protocol and software (Codenomicon, 2010).

2. Problems with Metrics

While the metrics currently available are useful to some degree, they have some problems that significantly reduce their effectiveness. Further, there are more problems with the theoretical underpinnings of computer security that make it difficult to provide meaningful metrics. This section will provide a brief introduction to some of these problems.

One of the major problems with providing metrics for computer security is choosing metrics that do not rely upon subjective "measurements". Deciding how critical a given application or system is for other systems or objectives is subjective. One person may consider a system to be more critical than another person may believe it to be. These differences may be a result of personal bias, a misunderstanding in the metrics, a difference in terminology, or just a difference in opinion. Subjective metrics are better than not having any metrics if the metric is clearly and well defined.

Many metrics are static. This makes the metrics easier to keep for longer periods of time without needing to invest more money to re-think or re-design the metric. Examples of these metrics include system and software complexity, certifications, and standard requirements. Loopholes or weaknesses can often be found in something that sticks around long enough (MITRE, 2002). Code can be produced that may score as being complex but actually be simple to a human being or vice-versa. Systems may be designed rigorously but also intentionally designed to be insecure, and systems proclaiming a certain status may be outdated and no longer secure but still appear so.

Another major difficulty in providing metrics is deciding on which metrics to provide. One of the more obvious metrics is a measurement of the amount of work or resources required to compromise a system. To construct this metric, we must be able to say that implementing a security mechanism X costs Y which will raise the cost of compromising the system by Z. With this kind of metric, costs can be compared making it easier to determine which mechanisms to procure (McHugh, Williams, & Skroch, 2000).

There are several issues with this. First, we often miscalculate the amount of work or the cost required to properly implement a security mechanism (Charette, 2005). We do not always understand the risk introduced by the addition of a new element to a system. Finding vulnerabilities and developing attacks seems almost random and, therefore, difficult to truly measure.

It is important to choose the right metric. If the metric is improperly designed or handled, several issues may arise. Too much information may be lost. Metrics may be

ineffective or misunderstood. Also, some may be readily available waiting for the right person to ask the right questions.

Composition is a significant issue with metrics (MITRE, 2002). Two smaller portions of a system may be measured and analyzed. When the two pieces are combined and evaluated, the composition may be difficult to analyze properly or provide unexpected results.

Some metrics are based on current knowledge to determine a pattern or trend. For example, a given application may exist for quite a while without any discovered vulnerabilities. One may think that this application is secure and does not provide much risk. While the trend may be consistent for a while, there is no firm way to tell when that trend may fail drastically. A slew of vulnerabilities may be discovered for the application that may prove too costly to fix properly. So, these types of metrics may indicate that a certain set of applications are more secure than another, but they cannot make any guarantees. For example, security patches to a set of software tend to fix vulnerabilities and not introduce new ones. There are several examples where this trend fails including a recent patch for Adobe Acrobat (Landesman, 2010).

3. Conclusions on Metrics

Metrics are useful. They must be continually studied for new metrics and to re-evaluate established metrics. However, there may always be metrics which require human analysis. Further, it takes the right humans to think of and design useful and appropriate metrics.

C. HEURISTICS

There are several heuristic methods that have been used to analyze the security properties of systems. These methods include: fault tree analysis and attack graph generation. These methods aid in systematically thinking about a system, gathering and organizing information about the system, and analyzing the information to determine the security strength of the system.

1. Fault Tree Analysis

There are two main approaches to logical reasoning: inductive and deductive. Inductive methods start with specific observations and work towards generalizing the information into theories. The inductive approach promotes exploratory methods, especially at the beginning (Trochim, 2006). In terms of Fault Tree Analysis, the inductive method assumes some initiating event and then determines what events may result from the initial event. For example, one may assume that a specific host has been compromised and then attempt to determine what may happen to the system given the host has been compromised.

Deductive reasoning starts with a theory or generalization and then works toward specifics. This approach tends to be more focused and narrow. It concerns itself with testing or confirming a hypothesis (Trochim, 2006). Fault Tree Analysis is an example of a deductive approach. In this analysis, a main failure state will be chosen and then chains of smaller failures that lead up to the main failure will be explored. This method is used to determine how a system may arrive at certain failure states.

While primarily deductive in nature, the use of Fault Tree Analysis should not imply an avoidance of inductive reasoning. Inductive reasoning may be used to provide new ideas or insight into the system or help reduce the system to important sections. Fault Tree Analysis can sometimes be lengthy and expensive. So, inductive reasoning may be used to help analyze certain aspects of the system in a faster and cheaper manner.

Fault Tree Analysis focuses on failure states and events that lead to such states for several reasons. It is not always clear when or if a certain system is in a successful state or will continue to be successful. However, failure states, while sometimes difficult to detect, are a little less subjective. For example, an airplane designed to do certain things may never meet everyone's expectations. If the airplane were to crash, it is obvious that something went wrong and the airplane failed (Vesely, Goldberg, Roberts, & Haasl, 1981). In the computer security realm, we may never be sure a system has not been compromised and is, therefore, successful. But, if we detect a breach of security, we know for sure the system has reached a failed state. Also, there are generally more successful states than failed ones,

so a reduced search space will help significantly.

A fault tree is not quantitative but qualitative. However, a fault tree may help facilitate the process of providing quantitative analysis (Vesely et al., 1981). Once a tree is created, probabilities or other measurements may be applied to each event on the tree if such information can be determined.

2. Attack Graph Generation

Attack graph generation is an inductive version of Fault Tree Analysis. This method starts in an initial state with the attacker not having compromised anything. The graph is then generated based on sequences of attacks the penetrator may conduct on the system. The method tests to see if there exists an attack path starting from a "proper" system state which leads to a node where the system has been compromised.

Attack graph generation provides several benefits. Many different attack paths compromising a specific property may be generated. This differs significantly from other model checkers which may never provide a counter example to the security property or only one at a time (Wing, 2007). Generating more than one attack path may reduce the number of iterations the model may need to be checked. Once a model of the network and other relevant information has been provided, the sequences of attacks on the network may be generated automatically. This reduces the number of errors that may be caused if created by hand. If done by hand, the attack graphs may be incomplete (i.e., missing attacks) or contain redundant or irrelevant paths (Wing, 2007).

Attack graph generation does have one significant drawback: it can only function on known attacks. It may be able to enumerate all known attacks and possible attack paths for a given system. However, it does not really have the capacity to theorize as to future attacks.

Graphs generated via this model provide some useful information for analysis. The existence of various attack graphs may be used to show how vulnerable a system is. It is not useful to spend millions of dollars protecting a system from zero-day attacks if it is still vulnerable to known attacks. The graphs may indicate functionality that should be added

or removed. Some of this functionality may be old and no longer used on a regular basis and may therefore be removed. New functionality may be added to provide another layer of security in the system (Wing, 2007).

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ARCHITECTURES AND ANALYSIS

This chapter focuses on systematically analyzing the added security provided by adding an additional layer of security. As an example of the approach, this chapter applies Fault Tree Analysis to assess the strength of the potential increase in security of a system that uses two layers of encryption instead of just one. Before being able to do this, several architectures are introduced. Architectures for both one layer of encryption and two layers are described. These architectures will help to provide a little more information and structure to the policy and make the analysis more complete and useful. In this chapter, we describe the strengths and weaknesses of each architecture to help indicate where they may be useful. To help focus the Fault Tree Analysis efforts onto one particular architecture, some quick analysis is applied to each architecture to determine which of them would be interesting to further study.

A. EXAMPLE SYSTEM

The example system is fairly simplistic. It consists of two or more protected networks. These networks are initialized and configured in similar manners. These protected networks communicate with each other over other networks, called unprotected networks, that are outside of the system's control. Every network employs basic communication protocols (e.g., TCP/IP protocols) necessary to communicate between the networks.

Some of the data upon which the networks work is sensitive and must be protected. The general policy introduced in Chapter I requires the data to be protected from being deduced or modified without proper authorization while traveling across unprotected networks. Since unprotected networks are outside of the system's control, the protection for the data must be applied before leaving the protected network by either the host or the gateway. These protected hosts and gateways may use one or more encryption mechanisms and protocols (e.g., SSH and IPsec) to satisfy this policy.

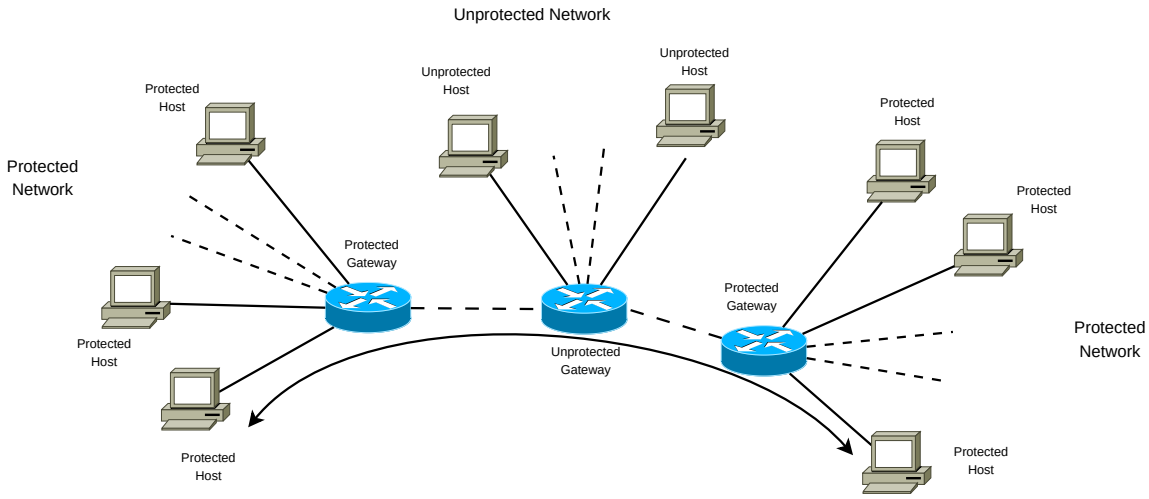

Figure 5: Pictorial representation of the example system.

B. ONE-LAYER ARCHITECTURES

One-layer architectures simply use one method of encryption at a time to protect network traffic. The encryption may originate and end at a host, a security gateway, or use some combination of the two. This basic architecture has several benefits. First, the system is significantly simpler than a multilayer architecture. As a result, it is easier to understand, evaluate, setup, and maintain. It is also easier to test, monitor, and ensure that it is working properly. This style of architecture is used often and is known to work well for many networks. However, this architecture does have some problems.

The most notable "problem" is that having only one layer of encryption means there is a single point of failure. If the encryption is compromised, then the security is compromised. Another issue is that, in some circumstances, having only one layer of encryption restricts the policy granularity.

A single-layer approach to encrypting network traffic lends towards two major categories of architectures: host-to-host and gateway-to-gateway. The benefits and consequences of these two types of architectures are discussed below.

1. Host-to-Host

A host-based architecture involves cryptographically secure communication channels between machines such as user workstations and servers. Some examples of ways to secure these types of connections include SSH tunnels, IPsec in transport mode, and a TSL connection.

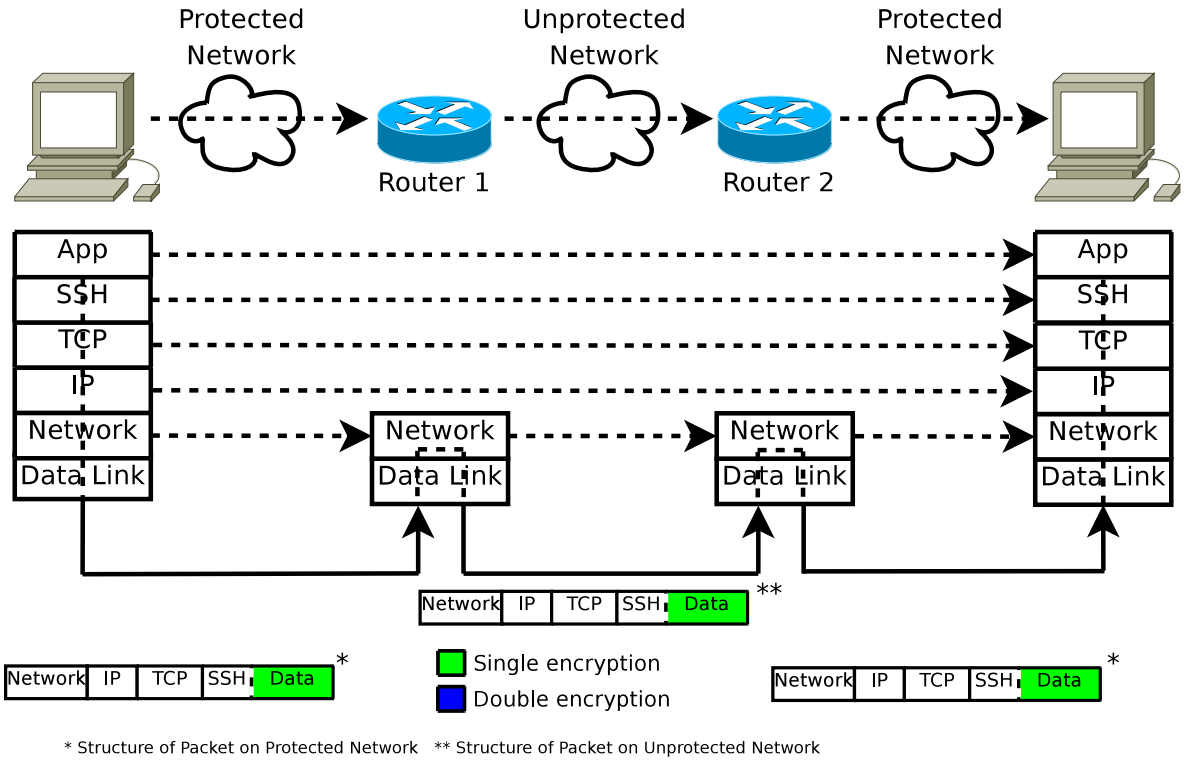

Figure 6: One layer of encryption provided by the hosts via SSH tunnels.

Protecting channels host-to-host has several benefits. A security policy can be applied more flexibly at the host level than at the gateway level. For example, one tunnel can be used to encrypt all traffic generated by the host or individual application-specific tunnels may be used. Host-to-host encryption protects network traffic throughout its journey across the network. If one host is compromised or goes down, the encryption between other host pairs will generally not be affected.

There are some drawbacks to host-based encryption schemes. These architectures will be more complicated than architectures that employ encryption tunnels between gate-

ways. In host-to-host encryption, every host must be properly setup, configured, maintained, tested, and evaluated. As a result, more work is required for the system than in a gateway-to-gateway architecture. Also, IP header information cannot be properly hidden unless the information is obfuscated via NAT'ing. The host cannot protect the headers because any new headers may still be associated with the host which defeats the purpose of hiding the original headers.

2. Gateway-to-Gateway

Network gateways receive network traffic bound to and from at least one machine, process some of the information, and then act appropriately such as forwarding the network traffic to the next destination. These gateways may be routers, network-based firewalls, network intrusion prevention or detection systems, or an encryption device for the network traffic. It does not matter on which device the encryption mechanism sits as long as all of the network traffic that must be encrypted or decrypted goes through the gateway appropriately. IPsec in tunnel mode is one of the most prominent types of encryption mechanisms that will work well in a gateway-based scheme.

Using one layer of encryption at the gateway level has a few benefits. It is a simple encryption scheme. It is easy to design, understand, evaluate, and test. Fewer components will have to be setup and configured properly. The least number of encryption keys must be created and maintained out of all the architectures presented. Important packet header information can be protected if the scheme encapsulates the original headers within new headers and encrypts the original headers. IPsec in tunnel mode will provide this kind of protection.

Gateway-to-gateway encryption does miss out on some useful features. Since the cryptography is done at the gateway, network traffic between the workstations and the gateways is not as well protected as host-based encryption. Hence, different properties about the physical security of the system may be required. It may be more difficult to attain the level of policy enforcement granularity desired when encryption is done at the gateway level. If the cryptography is some how lost or compromised at the gateway, all traffic going

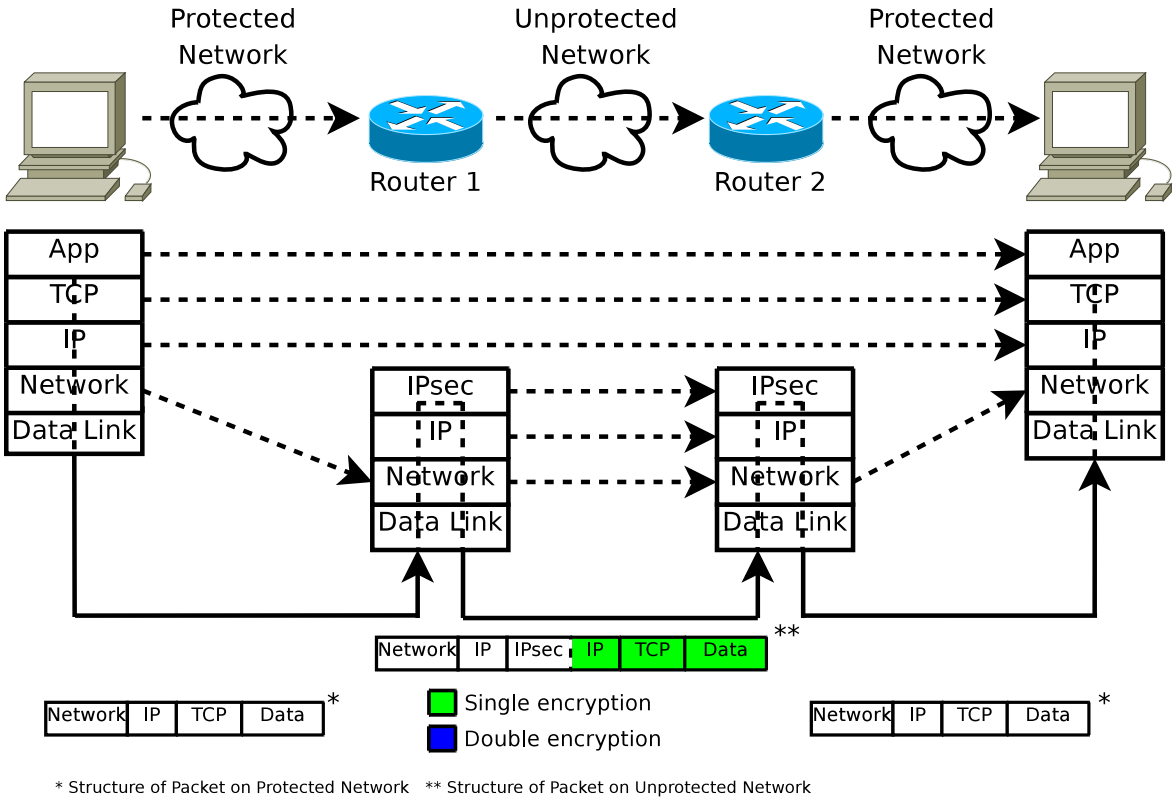

Figure 7: One layer of encryption provided by the gateways via IPsec tunnels.

in and out of that protected network may be compromised.

C. TWO-LAYER ARCHITECTURES

There are several reasons for using two layers of encryption. First, two layers of encryption may increase the assurance of having at least some amount of encryption on the data. If one of the layers of encryption were to somehow fail, the other layer may still be working. This has an advantage over one layer of encryption because if the single layer were to fail, there would be nothing to protect the confidentiality of the network traffic. Encrypting data more than once may increase the key space of the encryption and make it more difficult to brute force the encryption keys. Having two layers of encryption allows for more flexibility in the security policy of the system because more options are available.

The increased security that comes with this type of architecture does come with a cost. The system is significantly more complex than an architecture that only employs one encryption mechanism. This makes it more difficult to understand, setup and configure, and maintain the system. The increased complexity also makes the system harder to test and evaluate. Further, having two encryption mechanisms means more software is running on the system which inherently means an increased number of vulnerabilities and avenues of attack on the system.

1. Host-to-Host

One possible encryption architecture for the system involves doing both layers of encryption on each host machine. For example, the host could setup an encryption tunnel via IPsec in tunnel mode to protect the machines communications in general. It could then protect each individual application with SSH tunnels or a TLS connection.

This type of architecture has a few benefits. First, this architecture allows for more fine grained tunnels. These tunnels can be fine-tuned for just about anything. The tunnels can be associated with specific hosts and even specific applications. Having encryption provided by the host means encrypted data is protected throughout its journey once it leaves the host machine. This architecture also has the benefit of not having one central location

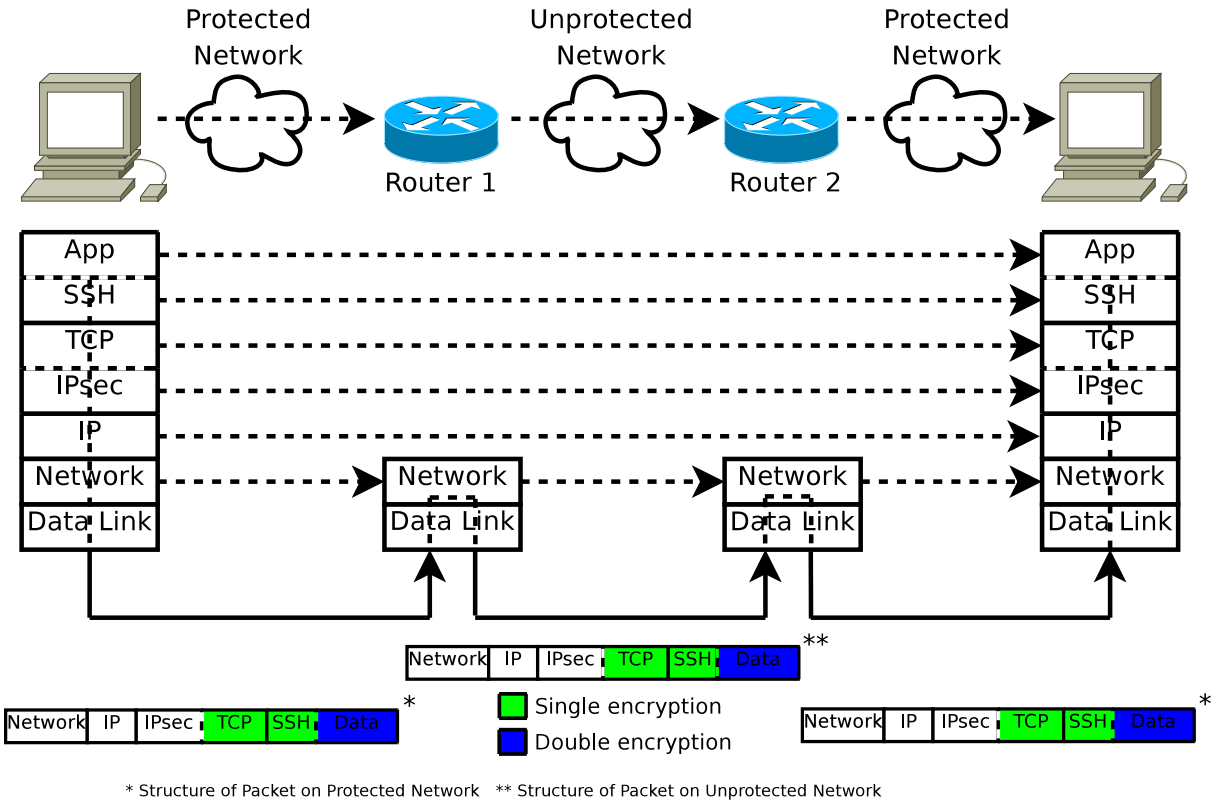

Figure 8: Two layers of encryption provided by the hosts via IPsec in transport mode and SSH tunnels.

for a layer of encryption. If an attacker wanted to decrypt all of the traffic on the network, he or she would have to attack each host on the network or brute force each key. This could significantly increase the amount of work an attacker would have to do. However, if an attacker does manage to break into one host, the attacker may not have to do much more work to gain access to the rest of the similarly configured hosts thereby decreasing the effectiveness of having host specific encryption.

This architecture has several issues, however. First, it provides for a single point of failure for the encryption of a particular host's traffic. If an attacker is able to break into the host machine, the network traffic produced by that machine will no longer be protected with two layers of encryption. Further, this architecture would be complex and require a significant amount of work to setup, configure, and maintain the system. Having both layers of encryption on the host will make it difficult to have strong network-based intrusion detection and protection. The traffic cannot be filtered well because the useful (i.e., original) headers are encrypted. As with the host-based, single-level encryption scheme, packet header information cannot be properly protected because any new headers that may be applied will still be associated with the host. This means that host packets will still be vulnerable to some traffic analysis attacks

2. Gateway-to-Gateway

Another architecture would encrypt traffic two different ways at the security gateway. For example, the gateway could have a tunnel setup between other gateways on the protected networks with IPsec in tunnel mode. It could have yet another instance of IPsec in tunnel mode or individual tunnels could be used for each host-to-host communication channel.

The encryption for the entire system would be more centrally handled by being done at the gateways instead of being done at each host. These tunnels would only have to be made between gateways on the edge of each protected network. Far fewer keys would need to be created and distributed than with the host-to-host architectures. Thus, this method may be significantly easier to implement and maintain than the first architecture. Host

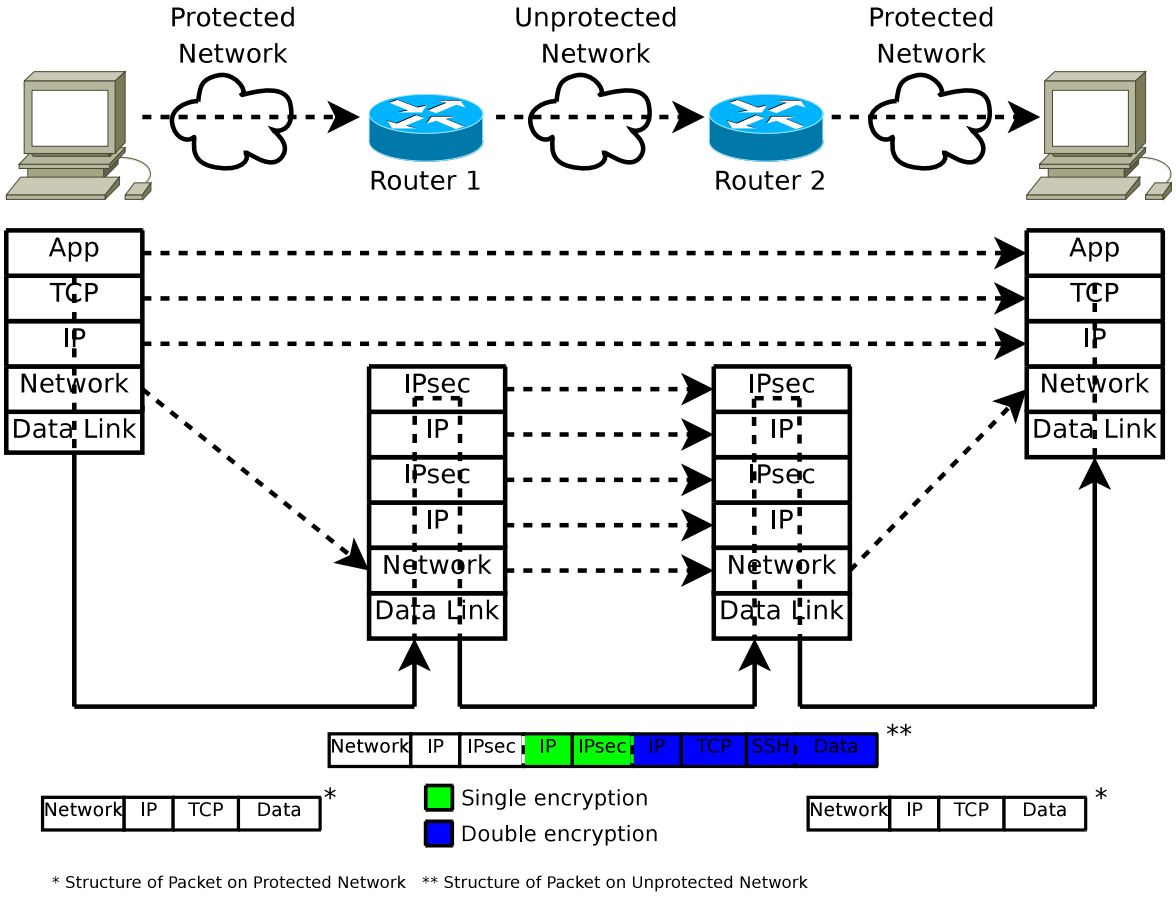

Figure 9: Two layers of encryption provided by the gateways via IPsec in tunneling mode.

packet header information can be protected if the headers are encapsulated within new headers that are associated with the gateway.

This architecture is similarly problematic as the host-to-host two-layer architecture in that it provides a single point of failure for the encryption. However, a failure at the gateway in this architecture has significantly more consequences than a failure at a single host. All network traffic will be exposed if the device is compromised. Having to both encrypt and decrypt traffic for a number of hosts will require a significant amount of processing power from the gateway. If the gateway does not have sufficient computing power, the system may suffer from a Denial-of-Service (DoS) "attack".

3. Host-to-Host and Gateway-to-Gateway

The best approach to two layers of encryption is probably to have one layer on the host machine and a second layer on a gateway or another device that specializes in encryption and decryption.

With both devices handling a separate layer of encryption, there is no single point of failure for the encryption. Also, this architecture provides for more flexibility and better organization than the other two-layer encryption schemes. Fewer keys and encryption tunnels must be maintained than the two-layer, host-to-host encryption architecture. Since the host provides one layer of encryption, all data is protected throughout its journey. The gateway can encapsulate the host's header information within new gateway headers and encrypt the old host headers helping to protect them from traffic analysis attacks.

This architecture does have some weaknesses. The system becomes more complex because both host machines and gateways must be configured and maintained properly. The system will have to maintain more keys than having two layers of encryption on the gateways.

D. QUICK HIGH-LEVEL ANALYSIS

As mentioned earlier, we will apply deductive reasoning and fault tree analysis to the problem at hand. However, inductive reasoning may be useful in quickly assessing

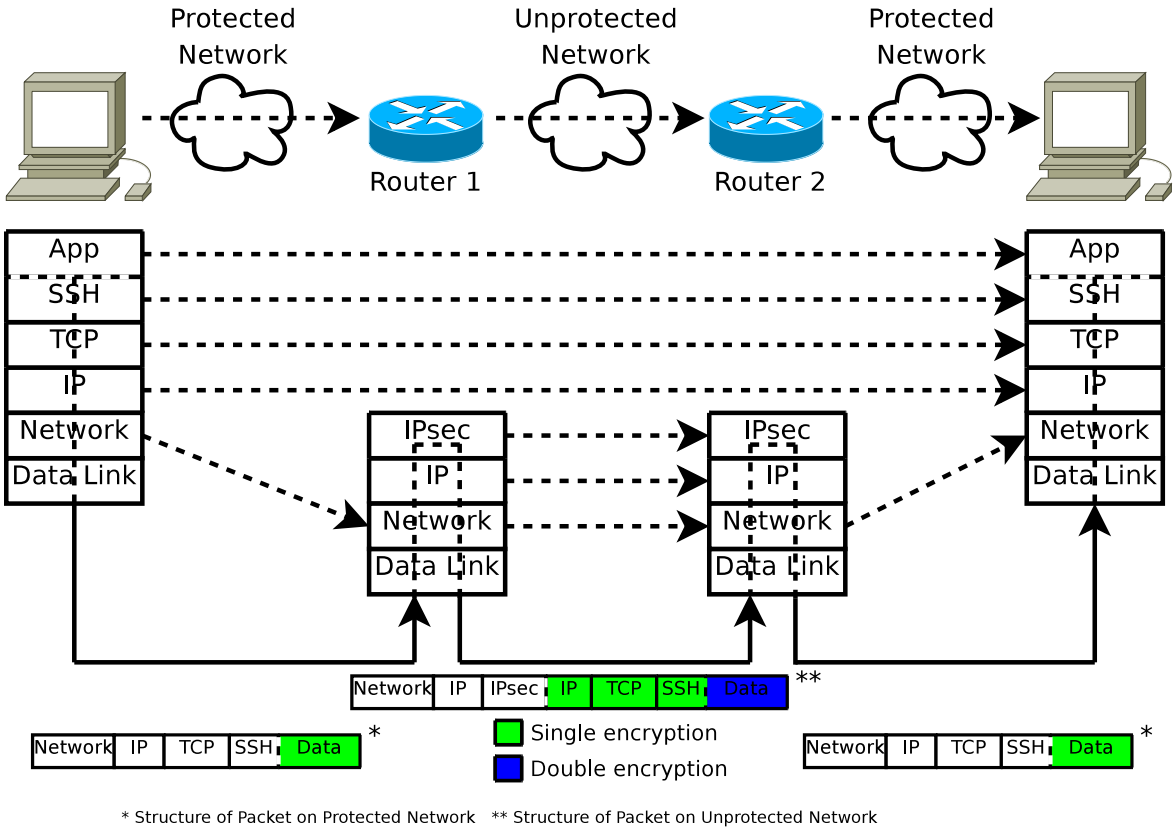

Figure 10: One layer of encryption provided by the hosts via SSH tunnels, and one layer of encryption provided by the gateways via IPsec in tunneling mode.

some weaknesses of the architectures. In this section, each architecture is assumed to have experienced some kind of failure: either a host that is sending or receiving data is compromised or a gateway is compromised. Asking what happens to the network and what network traffic is compromised after each initial failure will help indicate apparent weaknesses in the system.

If a host is compromised, the attacker has full access to the data that exists on that machine. As such, an attacker on any compromised host may access any network traffic originating from or arriving at the compromised host. If a gateway is providing a layer of encryption and the gateway is compromised, the attacker has access to the keys used by the encryption mechanism on the gateway and can access any traffic going through that encryption mechanism. If the gateway is providing both layers of encryption or the only layer of encryption, all traffic processed by the gateway is compromised. If another encryption is provided at another layer in the network (i.e., at the host level) and the gateway provides one as well, the encryption is weakened but not necessarily compromised if the gateway becomes compromised.

Architecture	**Host is Compromised**	**Gateway is Compromised**
Host	Host traffic compromised	No impact on cryptography
Gateway	Host traffic compromised	All network traffic compromised
Host + Gateway	Host traffic compromised	All cryptography for the network weakened
Host + Host	Host traffic compromised	No impact on cryptography
Gateway + Gateway	Host traffic compromised	All network traffic compromised

Table 1: A quick inductive analysis of affects on a system given a certain failure.

Of the architectures with two layers of encryption, only one architecture seems worth the extra effort: one layer at the host and the other at the gateway. If the architecture with two layers of encryption at the gateway has the gateway compromised, the entire network traffic is compromised. This must be avoided. The architecture with two layers at the host provides no significantly new functionality. Thus, we will focus on the architecture with encryption mechanisms at both the host level and the gateway level.

E. FAULT TREE ANALYSIS

In this section we analyze the architecture with one layer of encryption at the host level and the second at the gateway level. We start with a failure state or the loss of a security property and then deduce paths that will lead to that failure state. The security property in question applies to the two layers of encryption architecture with one layer at the host and one provided by the gateway. The failure state chosen is a loss of confidentiality in data that should be protected (e.g., the data is compromised).

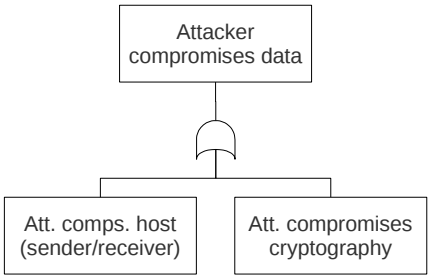

Figure 11: Two main ways to access protected data: access it when not protected or compromise the protection.

There are two main ways to access data that is intended to be protected. The first way is to gain access to the data when it is not protected by taking control of a host machine that is either sending or receiving the data. Since the host is most likely going to work on unprotected data, an attacker should be able to access the unprotected data once he or she has gained sufficient privileges on the host machine. The second way is to compromise the cryptography used to protect that data. With the cryptography compromised, an attacker can access the protected network traffic.

1. Host Compromised

An attacker can compromise one of the sending or receiving hosts in two different manners: from within the network or from outside the network. This thesis does not explore the idea of an attack from inside the network. An inside threat is difficult to analyze without more information about the network. Also, an insider often has significantly more

Attacker compromises data:

1. Attacker compromises sending or receiving host.
   (a) Attack originates from inside the network.
   (b) Attacker compromises host from outside the network.
   Note: Attacker must gain access to host prior to attacking the host.
      i. Attacker gains access to host from outside the network.
         A. Attacker has access because the gateway does not provide protection.
         B. Attacker gains access to network by subverting gateway's security.
      ii. Attack originates from outside the network.
         A. Attacker exploits a vulnerability in the OS.
         B. Attacker exploits a vulnerability in the encryption software.
         C. Attacker exploits a vulnerability in the key distribution or management software.
2. Attacker compromises cryptography protecting data.
   (a) Attacker discovers the encryption keys.
      i. Attack originates from inside the network.
      ii. Attack originates from outside the network.
         A. Attacker takes advantage of a weakness in the key distribution software.
         B. Attacker takes advantage of a weakness in the key distribution protocol.
         C. Attacker compromises the key server (if one exists).
         D. Attacker takes advantage of a weakness in the encryption/tunneling protocol.
         E. Attacker uses brute force to determine the keys.
   (b) Attacker takes advantage of unintended services to reveal plain text
      i. Attacker takes advantage of a weakness in the configuration of the encryption/tunneling software.
      ii. Attacker takes advantage of a weakness in the cryptographic communication protocol.
      iii. Attacker takes advantage of a weakness in other protocols functioning within the system.

Table 2: Summary of fault tree.

privileges and can access more than an attacker outside the network. So, more detailed analysis, while useful, may not be as beneficial as studying an outside threat in the general case.

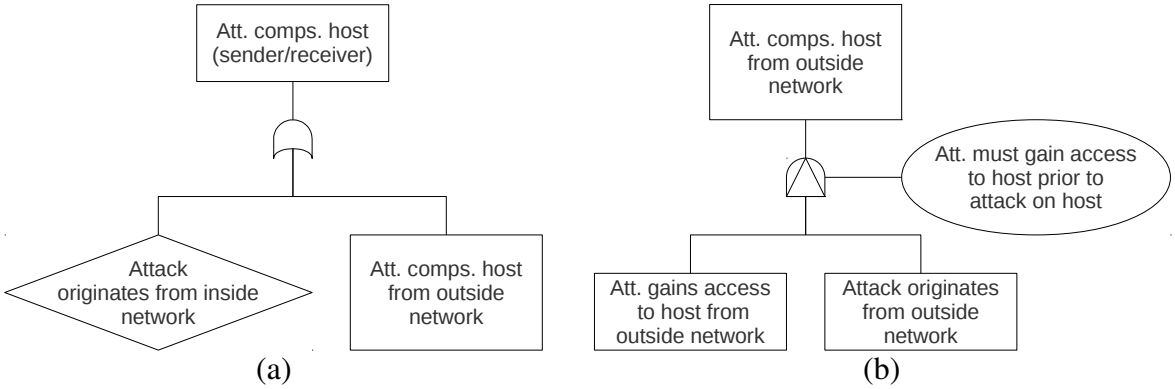

Figure 12: (a) There are two ways to compromise the host. (b) The attacker must gain access to the host first.

To compromise the host from outside of the network, an attacker must first gain access to the host. Depending on the configuration of the network and the security gateway, the gateway may be providing protection to the hosts on the network by restricting outside access. If this is not the case, the attacker already has access to the host. The gateway may not be restricting access to the host for two main reasons: the gateway has been improperly configured or it was designed to allow some traffic through to the hosts.

If the gateway does restrict access to the hosts, the attacker must take more steps to gain access to the host (Fig. 14). The attacker must somehow subvert the security of the gateway by posing as a trusted host, tricking the gateway into rerouting traffic, or by breaking into the machine and changing permissions such that he or she can access the host. An attacker has several possible paths including: spoofing his or her IP address, modifying routing tables, or exploiting one or more vulnerabilities provided by the OS, encryption/tunneling software, and key distribution software running on the gateway. Several examples of such vulnerabilities are provided in the Appendix. While most if not all of these vulnerabilities have fixes available and may no longer be a specific concern, they do serve to show that such exploits are possible and must not be deemed impossible.

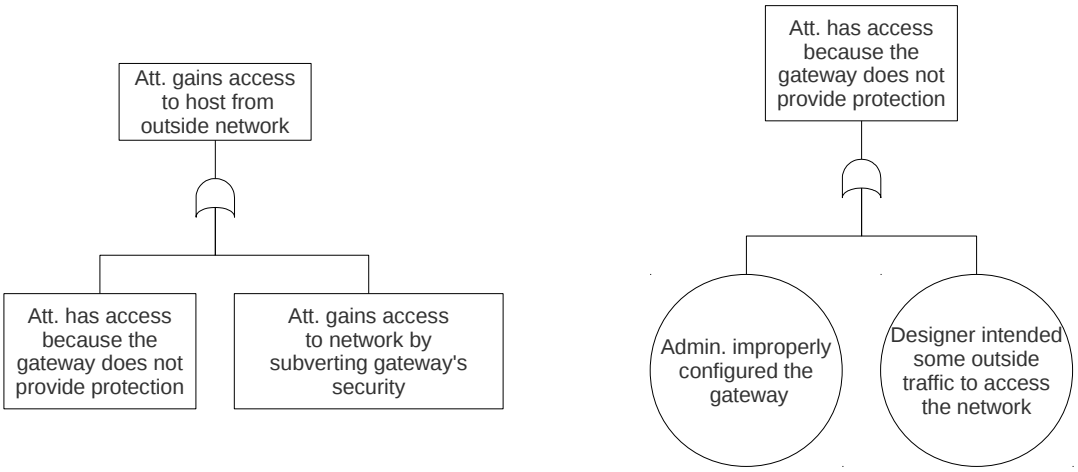

Figure 13: (a) There are two ways to gain access to the host. (b) The host may not be protected by the gateway.

Once the attacker has gained access to the host, he or she must gain privileges on the host. The attacker must exploit one or more available vulnerabilities on the host if he or she does not know appropriate user credentials. These vulnerabilities may arise from software running on the system that would have been there without our policy of requiring encryption and tunneling software (e.g., the OS and other services that may reside on the host) or software added because of the policy (e.g., encryption, tunneling, and key management software).

Some of the software and protocols used on the host may be very similar or exactly the same as those used on the gateway. This may cause issues in that these similarities may include similar vulnerabilities which could make it easier for an attacker to gain access to the host's data. This issue of independence or lack thereof will be discussed in further detail later.

With access to the host and sufficient privileges on the host, an attacker may access the data we have been attempting to protect. This is obviously a failure in holding to our policy.

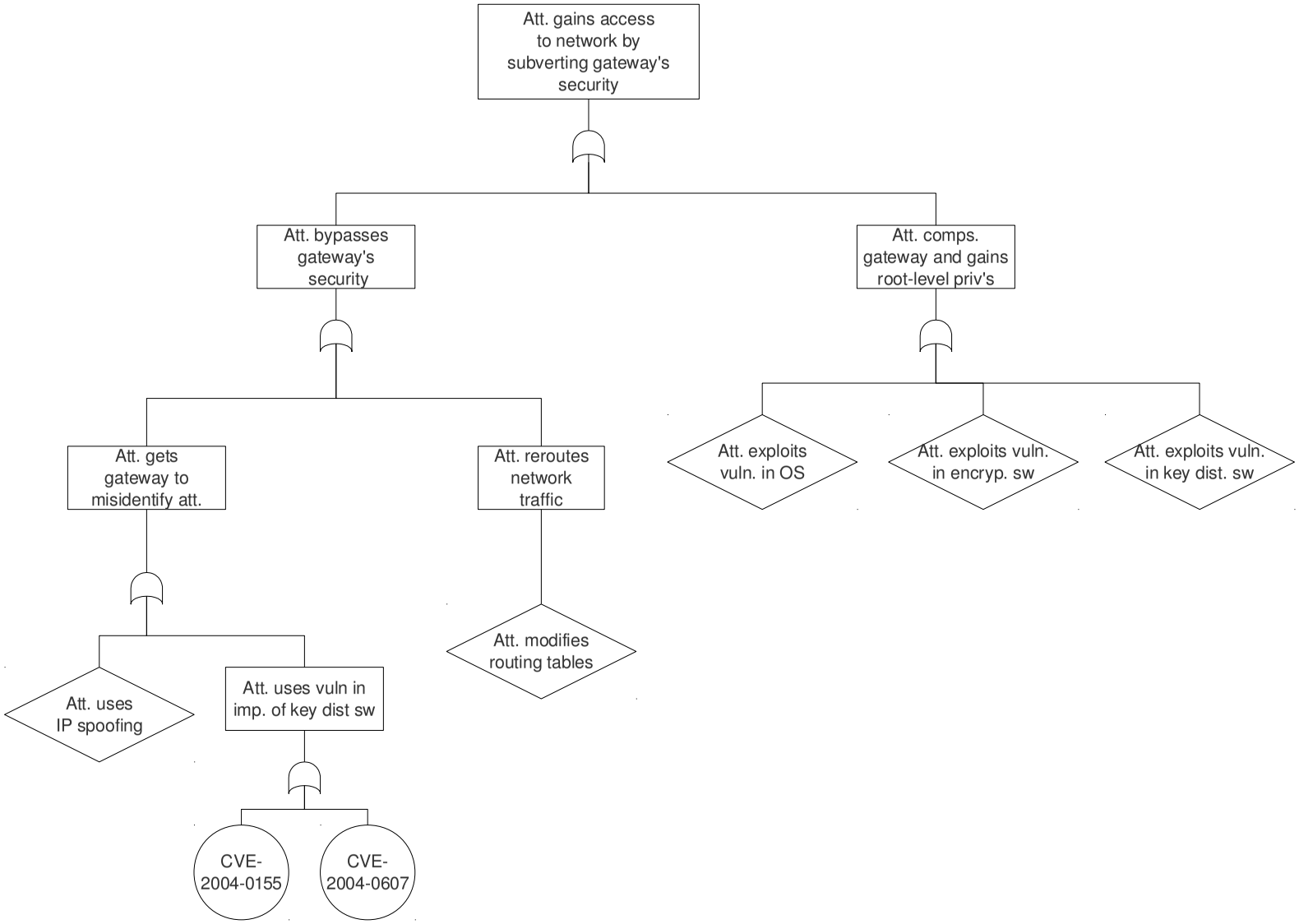

Figure 14: Gateway security must be subverted.

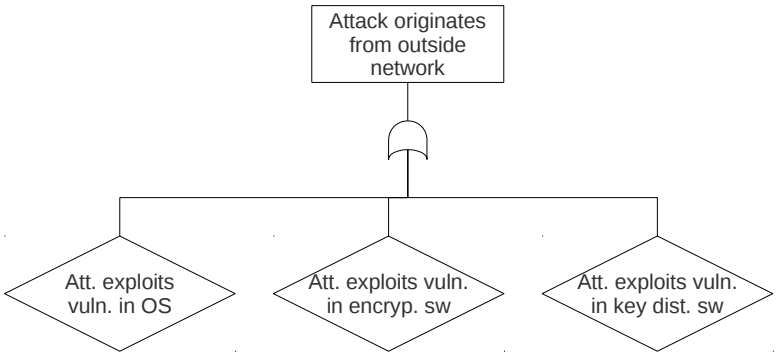

Figure 15: Attacker compromises the host.

2. Cryptography Compromised

This thesis outlines two general ways to compromise the cryptography of the system. One way is to discover the cryptographic keys used in the system. The second is to take advantage of unintended services provided by the system.

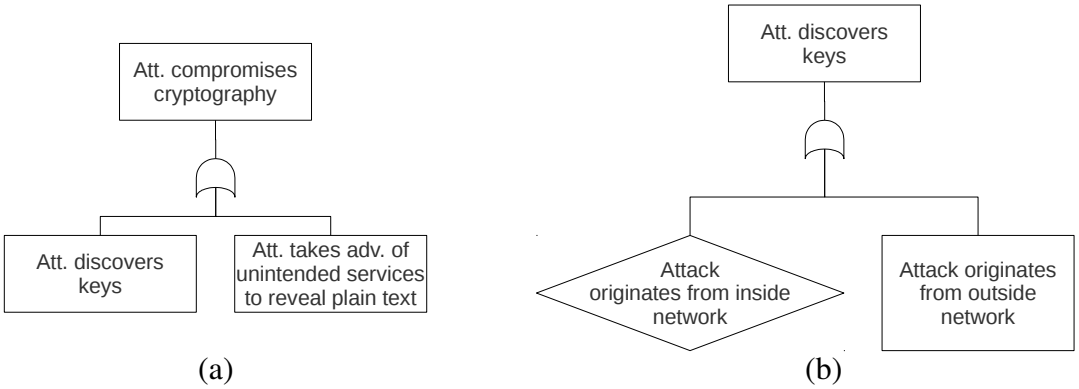

Figure 16: (a) Attacker compromises the cryptography protecting the data. (b) Attacker discovers the keys used to encrypt the data.

The keys may be discovered in two general fashions: from within or from without the network. As with the hosts, this thesis does not investigate the paths an attacker may take to obtain keys from the inside. An attacker from outside the network still has a number of possible ways to get the necessary keys.

The attacker may take advantage of weaknesses in the key distribution software or protocol, the encryption/tunneling protocols or by attacking the key server (if one exists). If the attacker were to compromise the key server, he or she may be able to generate new

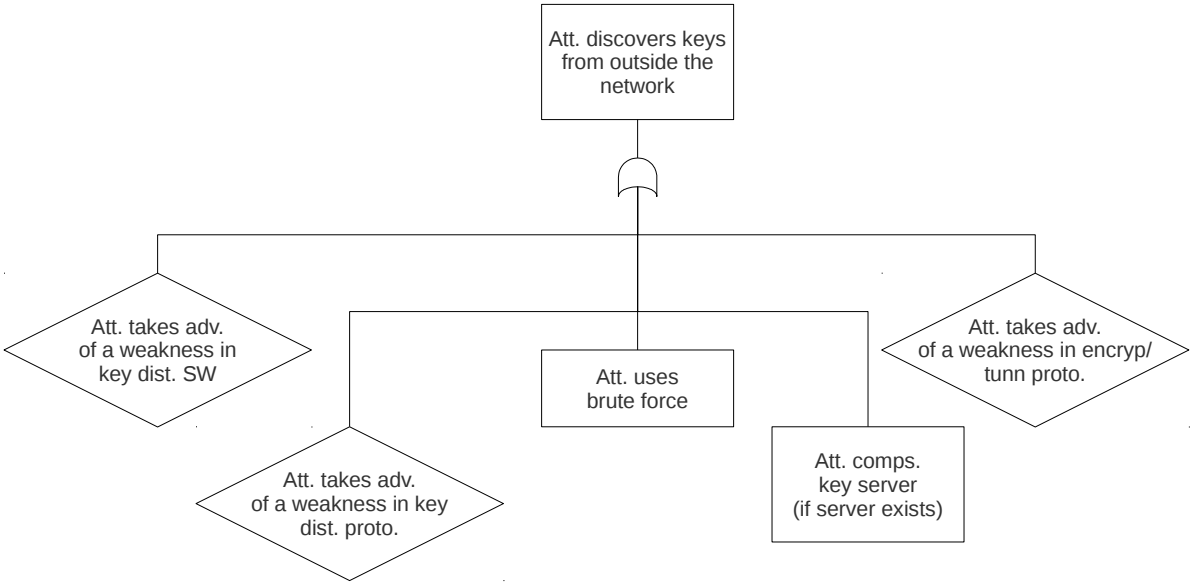

Figure 17: Attacker discovers encryption keys from outside the network.

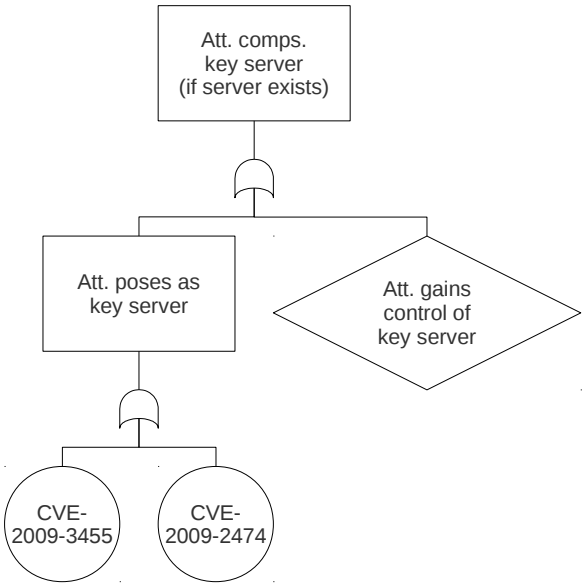

Figure 18: Attacker compromises the key server.

keys and distribute them for use throughout the network or recover keys currently in use or that have been previously used to decrypt captured traffic. The attacker has another method to discover the cryptographic keys: brute force.

Cryptographic keys are intended to take more work and effort to brute force than the data is worth. There are several things that may reduce the amount of work required to brute force the keys (Fig. 19). Weaknesses in the encryption algorithm or the implementation of the encryption algorithm may help indicate which keys are more likely than others effectively reducing the key search space. Pseudo-random number generators (PRNGs) not working appropriately may reduce the key space for the system. Also, the configurations for software involved in the encryption process may further weaken the cryptographic strength of the system by using weaker algorithms, smaller key sizes, or vulnerable protocols.

An attacker may be able to take advantage of unintended services in several different areas throughout the system. Some of these unintended services may provide a decryption service to unauthorized personnel thereby revealing plaintext to the attacker. He or she may be able to take advantages of weaknesses in the configuration of encryption and tunneling software implementations (Paterson & Yau, 2006). The attacker may be able to take advantage of a weakness in cryptographic protocols. For example, the original Needham-Schroeder protocol (Needham & Schroeder, 1978) allows an attacker to do such a thing by interacting with two normal clients in a certain manner (Lowe, 1996). The attacker may also be able to take advantage of other protocols on the system like ICMP to reveal part of or the entire plaintext of an encrypted message.

In this thesis, we provide a starting place for enumerating the different paths of events that ultimately lead to compromised data within the system. Computer security is very complicated and the different methods discussed above may be used in a number of different fashions, combinations, and orders. An attacker may be able to take exploit vulnerabilities in one or more machines to obtain information that will then help reduce the key space of the system. The attacker may be able to discover one key in one fashion and

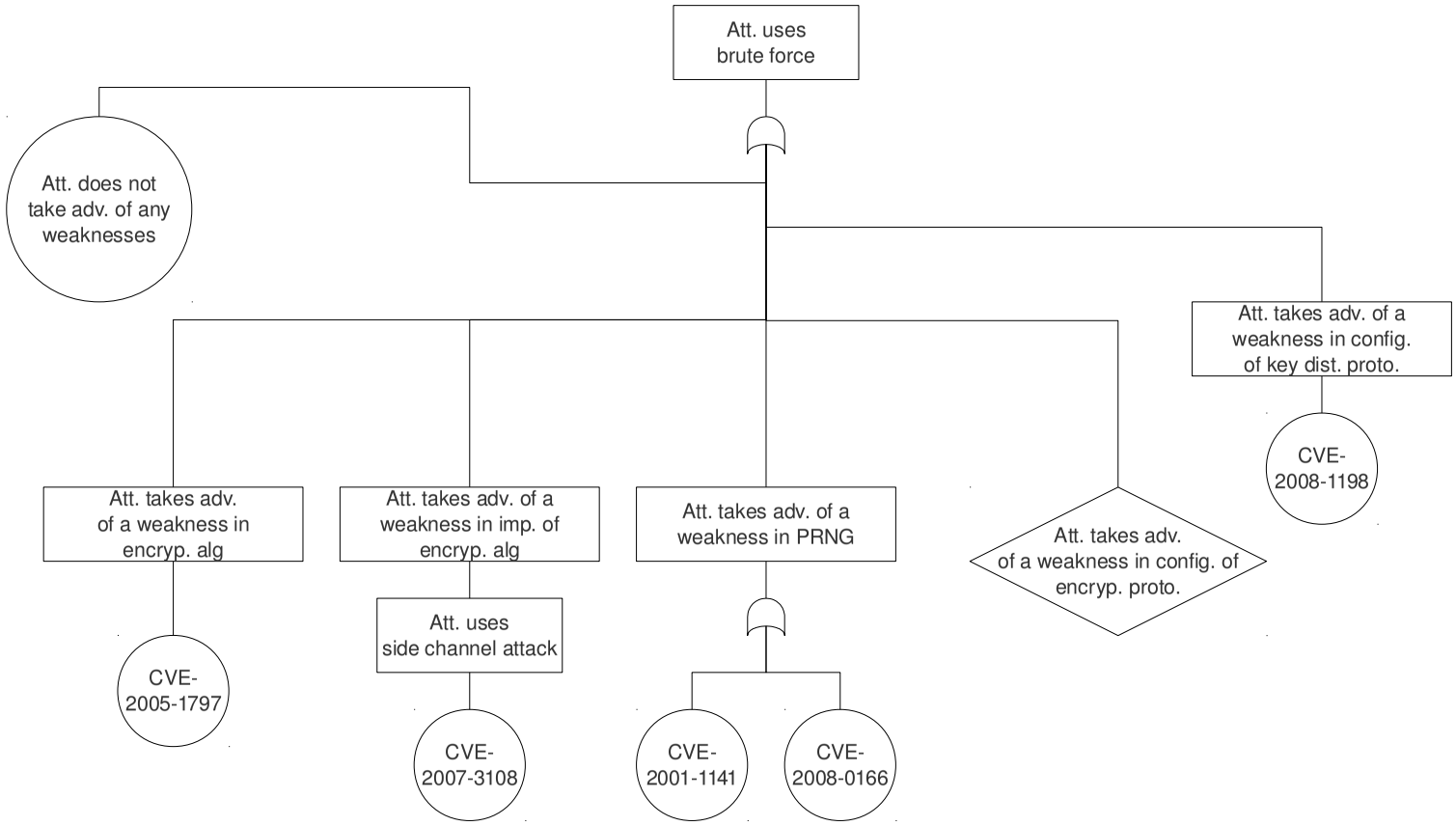

Figure 19: Attacker uses brute force to discover keys.

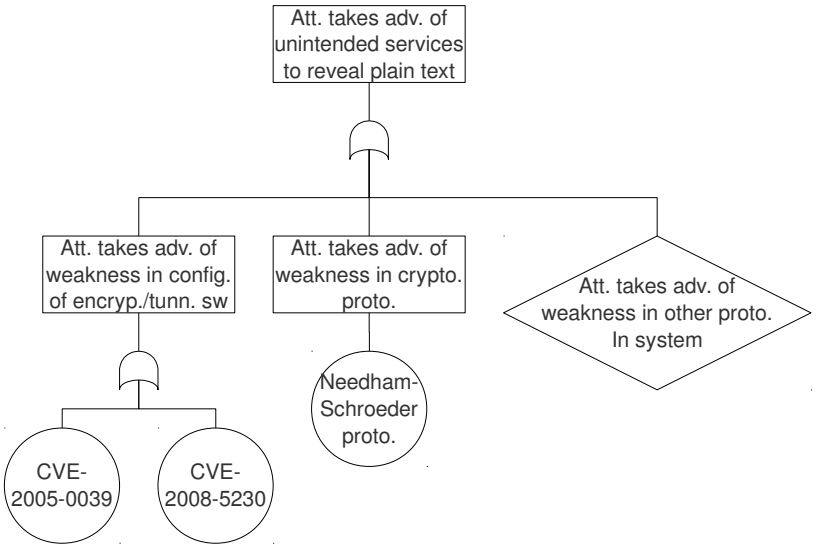

Figure 20: Attacker takes advantage of unintended services within the system.

then use an entirely different method to procure the other key. Many more types of attacks exist than those that have been presented in this thesis. The fault tree provided has been kept simple due to time constraints.

F. PROBABILITIES

Fault trees allow for a nice way of analyzing a system: the assignment of probabilities to failures of certain items or events. The hierarchical structure of a tree and the use of AND-gates, OR-gates, and other similar gates greatly aids this effort. Probabilities are assigned starting at the leaf nodes and then work up towards the root. Thus, a good understanding of the probability of certain events will be obtained after the tree has been analyzed.

Unfortunately, being able to apply useful probabilities to a fault tree in the field of computer security is very difficult. Specific physical components may be tested over and over again to roughly determine when a physical part may fail, and probabilities may be drawn from that data. It is much more difficult to determine when or if a vulnerability may be discovered for a certain set of software or protocols.

Fault tree analysis is still useful even if the majority of the tree is not filled in

with probabilities. First, constructing the tree for the system will help the analysts better understand the system and potential areas for attacks. Second, an analysis of the structure of the tree may help gauge potential strengths and weaknesses of the system. Third, other information may be associated with the tree. Other useful information that may be applied to fault trees will be mentioned in a later section. Finally, the tree may help bring to light portions of the system that depend upon other portions. Dependencies within a system are very important to computer security and will be discussed further in a later section.

G. OTHER USEFUL INFORMATION

According to the *Fault Tree Handbook* (Vesely et al., 1981), the nodes were intended to be filled in with probabilities. Probabilities for many events related to computer security are difficult to determine. Other numbers and pieces of information may be easier to provide and associate with these types of fault trees.

General indicators of possibilities may be useful. For example, we can say that a given event may be possible or impossible (Schneier, 1999). In a sufficiently complex environment, it may be difficult to state an event is truly impossible. These values may not be correct but they at least give a general idea as to the possibility of certain events. Instead of possible or impossible, other values may include highly likely, likely, unlikely, highly unlikely, or unknown.

Cost applied to the fault tree nodes may also be useful (Schneier, 1999). These cost values may include: the cost of hardware or software, man-hours, or the cost to bribe a janitor. General indicators may indicate the size of organizations that can implement such an attack such as individual, terrorist organization, or nation.

Other types of information may be applied here. A note can be used to indicate the action that makes an event impossible or explains other given values. For example, an attack may no longer be viable if a certain patch is applied to the system or a firewall with a given rule is implemented. This information could also indicate equipment that may be needed for the attack (Schneier, 1999). Even partial information that may prove useful

when other information is provided or the information is analyzed in a different manner.

Each node on the tree may contain any number or combination of the discussed information (Schneier, 1999). For example, costs and indications of possibility may be used. If the cheapest attack on the tree is shown to be impossible, efforts may be focused on other aspects of the tree or system.

H. DEPENDENCE

Dependencies, single points of failure, and defense-in-depth are all tied together. If we depend upon something, that something is usually important to the normal functioning of our lives. At times, we may tend to or need to assume that which we depend upon will be available or trustworthy when we need it. If our assumptions prove to be false, we may be hurt, ruined, or annoyed. In the case of computer security, dependencies and assumptions are often related to the safety of the system.

Single points of failure are very valid sources of concern when considering computer security and the requirements of the system. Aspects of the system that depend upon the same thing may all fail or may become vulnerable to attack if the underlying thing they rely upon fails or becomes compromised. For example, a network that relies heavily on access to the Internet may easily lose availability if it only has one line of connection to the outside world. An additional connection to the Internet may help meet the requirements of the system and avoid a single point of failure.

Defense-in-depth attempts to protect a system with multiple layers of defense. The idea is similar to having multiple walls, a moat, and soldiers protecting a castle. If one line of defense falls, the others should still be functioning and raise the cost to the attacker. The defenses are meant to hamper the attacker long enough for help to arrive, to tire and frustrate the attacker enough to give up, or to defeat the attacker. For the defenses to be effective, they must be independent. Defenses that depend on something in common may fall at the same time if the thing in common is compromised. For example, let us assume the attackers have cut off the castle's supply of water. With no refreshing source of water,

the moat will become less effective. Similarly, the soldiers will become less effective if they die of dehydration.

Computer systems are extremely complex, making it difficult to identify dependencies within the system. There are so many aspects to the system: the physical connections of the network, the logical connections and information flow of the network, the software running on each computer, hardware devices, protocols, libraries, and so much more. This section of the thesis introduces and briefly discusses some of the dependencies or potential dependencies of a system with one layer of encryption provided by the host and another by the gateway.

One of the most obvious of dependencies is the running operating system of both the host and the gateway. If the two OS's are similar and are running similar software, the machines run a risk of having a common vulnerability. In this case, an exploit used against the gateway may also be used against the host. This issue may be mitigated by having different OS's running on the gateway and the host.

The system's security depends upon the implementation of the encryption and encapsulation software (e.g., IPsec). If there is a vulnerability in the software, an attacker can exploit the system and gain access to the data we are trying to protect. If the same software is running on both the host and the gateway, they will share the same vulnerabilities. An obvious solution to this single point of failure is to use two different software packages.

Two seemingly different software packages may still share some dependencies. The software packages may have a different user interface but actually share the same core or back-end. For example, FreeBSD and OpenBSD use the KAME implementation of IPsec (http://www.kame.net/), and the native linux kernel implementation is a rewrite of the KAME project (Rosen, 2008). The suite of tools called IPsec-tools is derived from the KAME project and is used in many linux distributions (http://ipsec-tools.sourceforge.net/).

Different software packages may share other similarities. Software developers may have similar development principles and techniques (or lack thereof such as not properly sanitizing user input). Many people follow available tutorials. The developers may even

copy and paste code from other sources. Any of the above similarities may introduce common mistakes and vulnerabilities. Further, software developed using the same language may use similar libraries. These libraries may have vulnerabilities in them (e.g., http://www.juniper.net/security/auto/vulnerabilities/vuln3732.html).

Finally, software implementations depend heavily on the protocol upon which they are based. If there is a vulnerability in the protocol and the implementation follows the protocol, the implementation will exhibit the vulnerability. Vulnerabilities at the protocol level are an important illustration of dependencies because they are implementation independent (assuming the implementations follow the protocol sufficiently). Despite different implementations, a vulnerability in the IPsec protocol itself may leave both layers of defense vulnerable rendering the intended defense-in-depth less effective.

Cryptography typically relies upon several things including keys, pseudo-randomly generated numbers (e.g., IVs), and key distribution and management. Cryptography depends heavily upon keeping important information (e.g., keys) secret. If the right keys become known to the wrong people, the confidentiality of the system's data may become compromised. If the PRNG is not functioning properly, the key space may become significantly diminished (CVE-2008-0166) or make it easier to guess the pattern of the "random" numbers generated (CVE-2001-1141). If keys are not handled properly, unauthorized keys may be used or authorized keys may become available to unauthorized beings.

The entire system depends upon the administration of the system. All of the software on the system must be updated (or not), configured (properly or otherwise), installed or removed. Vulnerabilities may easily be introduced in this fashion. If the administrators are overburdened or do not have sufficient experience or the system is too complex, the system may be configured or administered improperly. Further, an attacker may reconfigure all of the machines containing data to grant access to the attacker if he or she has compromised the administration aspect of the system.

If one looks close enough, a single point of failure can be found for some portion of the system. For example, we discussed earlier about using a OS for the gateways and

a different one for the hosts. This does avoid a single point of failure regarding OS's at the system level. However, all of the hosts may still have the same operating system. A vulnerability found on one host will almost certainly exist on the other hosts making it easy to compromise many hosts on the network.

These hosts (and gateways) are not independent within their subsystem until each machine utilizes a fundamentally different OS. This is impossible for anything but a small network. Eventually, at least one OS will be installed on more than one host increasing the possibility of a common vulnerability being found.

This idea can be applied in several more areas including protocols for the encryption, encapsulation, and key management; cryptographic algorithms; and software implementations of the protocols and algorithms. The protocols that handle the encryption and encapsulation will pose an unavoidable problem. Entities attempting to communicate with each other must use the same protocol. With this in mind, let us assume that one protocol will be used at the gateway level (e.g., IPsec) and another protocol used at the host level (e.g., SSH). It would become incredibly complex to configure and manage if we were to require multiple protocols at the host level. The key distribution and key management protocols have similar issues.

There are many encryption algorithms with which we can work, but few are sufficiently secure for protecting important data. The encryption algorithm faces similar issues to the protocols already mentioned. For one host to communicate effectively with another, they must use the same encryption algorithm.

The implementations of the protocols and algorithms discussed above give us a little more flexibility. One implementation of a protocol may interact with a different implementation assuming both follow the protocol sufficiently. Multiple implementations of IPsec, for example, exist and may be used. This still does not rule out the possibilities of common dependencies or vulnerabilities as discussed earlier in this section.

I. MULTIPLE ENCRYPTION

As mentioned in the above section, it is important to understand or attempt to understand the dependencies between things within a system. This section will briefly explore the inter-dependencies between the two layers of encryption.

Computer security practitioners began doubly encrypting data with two different keys. They had intended it to make the two keys significantly more difficult to brute force than one key of the original size. By doing this, they had hoped to not have to modify or introduce any new algorithms or key sizes. They simply had to use the existing infrastructure in a slightly new way. Some expected to increase the key search space from 2^n with an n-bit key to 2^{2n}. As mentioned in the background, this is not the case. The key space can be reduced to 2^{n+1} or even smaller with the use of some other optimizations.

Since the meet-in-the-middle attack reduces the key search space to just twice that of a single key, it makes it look like the two encryptions are separate. The intent of double encryption was to "double" the key length and, therefore, increase the key search space to 2^{2n}. If that had worked, it would have meant that the two separate defense mechanisms were actually one solid and stronger defense mechanism. Instead, we got two roughly independent but similar defense mechanisms which only double the amount of work.

Working off the analogy of a castle defense, suppose an attacker takes 10 minutes to scale a wall and there are two walls with no defenders inside the castle. It would take an attacker twenty minutes to gain entrance to the castle. Similarly, if it took one attacker 10 minutes to dispatch one defender and there are two defenders, it would take the attacker 20 minutes to compromise the castle. However, if there is one wall and a number of defenders defending the wall and the castle, the defenses would exponentially increase the amount of time it would take the attacker to storm the castle. The defenders and the wall work together almost as a single unit to protect themselves so they work more effectively to protect the castle. Two walls do not do much of anything without other defenses. Two layers of encryption act much like the two walls. They provide a little more protection but not much more at all.

To tie it back into thinking about dependencies, having two walls is like having to independent defenses. These independent defenses are functionally similar and only slow the attacker for a marginal amount of time. Having walls and defenders introduces an interdependency between the defenses. The defenders rely upon the wall to limit the attacks that can be made, and the wall relies upon the defenders to help keep the wall functioning (or erect) and to throw the attackers back. These defenses rely upon each other, but they also provide different functionality.

J. COMPLEXITY

One of the biggest problems that arises from the use of two layers of encryption is the increased complexity of the system. More software must be setup, configured, and managed properly. More services will be running at one time. An increased number of keys must be created, managed, and distributed. More configuration options must be set and tested. More tests must be conducted. More time, money, and man-power must be spent to properly test, evaluate, and maintain the security of the system. This additional complexity may cause significantly more problems for the system and potentially increases the attack surface.

The complexity of the system can be increased even further by trying to provide truly independent layers of defense. Multiple OS's, encryption and encapsulation protocols, key distribution protocols, encryption algorithms, and software implementations of these protocols and algorithms may be used in a system to avoid a single point of failure. Adding in different protocols and software significantly increase the complexity of the system. They increase the quantity of things such as configuration options, and they raise the complexity of the logical design of the network. More things to keep track of make it difficult for a designer, installer, or maintainer to do their job properly. If there is any confusion as to the structure of the system, vulnerabilities may be introduced at many levels. Also, more software on the system means an increased number of vulnerabilities or potential vulnerabilities.

There may be more unforeseen issues having to do with the composition of protocols and the use of multiple encryption. A protocol may work well and be secure on its own. That does not indicate that the protocol does not become weaker or provide a vulnerability when combined with another protocol. We have a well known example of combining a known functioning protocol with itself, TCP over TCP. TCP is used all the time and works well. When combined with another instance of itself, problems can arise and the quality of the connection may degrade (Honda, Ohsaki, Imase, Ishizuka, & Murayama, 2005). A similar thing may happen with multiple encryption.

Simple systems, generally, produce simple failures that are simple to avoid or fix. Complex systems produce both simple and complex failures. These failures may be significantly more difficult to find, understand, and fix due to the complexity of the system. In order to really fix a problem, a good understanding of the system and the problem are necessary. A complex system is difficult to understand making it hard to find and fix the problem.

V. CONCLUSIONS AND RECOMMENDATIONS

This chapter mentions again the intended or hoped-for benefits of using two layers of encryption discussed in Chapter I. Relevant information pertaining to each of these benefits are quickly reviewed. The focus of the chapter resides upon Fault Tree Analysis and the potential usefulness of having a "backup" layer of encryption. This chapter also provides areas of future work and research. The thesis concludes with a discussion on costs versus benefits and provides some recommendations pertaining to the different architectures and their uses.

A. BENEFITS

As mentioned in Chapter I, some might think that using two layers of encryption provides significantly more protection over the use of one layer. The scheme should provide more network security by making it difficult to inject or read traffic by unauthorized users. The two layers should make it more difficult to perform traffic analysis attacks by hiding host header information. Multiple encryption should markedly increase the cryptographic strength of the system. Finally, one of the more important reasons for wanting two layers of encryption is the "backup" layer of protection the extra would provide were one to fail.

Two layers of encryption should provide more security at the network level. The data is already encrypted by the host. So, the network level encryption does not provide any new functionality for protecting the confidentiality of the data. It does provide a way of authenticating network traffic which could be useful for certain networks. Without the proper keys, an attacker cannot easily inject traffic onto the network. He or she must use inappropriate means to do so. This may make it more difficult to attack the network in general. This benefit is reduced in many networks that require access to unprotected networks. If this is the only benefit desired, the host layer of encryption is superfluous. The network layer of encryption will both protect the data when traveling across unprotected

networks and provide network level authentication.

The two layers of encryption should better protect packet header information of hosts on the network from those outside of the protected networks. The network level of encryption must encapsulate the host's header information, generate new headers, and encrypt the old ones. The host level of encryption is again unnecessary for this benefit. Also, some form of NAT'ing would provide similar protection though not necessarily to the same degree.

Multiple encryption increases the cryptographic strength of the system. This is generally true. As discussed in earlier chapters, the benefits of multiple encryption are not always as expected. Double encryption at most doubles the amount of work needed to brute force the keys. Single encryption is still viable. It is extremely difficult to brute force a key with a sufficient length of a well-tested and approved algorithm such as AES (Schneier, 2009). The NSA still allows the use of AES to protect classified information (CNSS, 2003).

Probably the main reason for using two layers of encryption is to have a backup layer of protection in case one layer were to fail or become compromised. Chapter IV explored this topic from the standpoint of dependencies. It is important to understand the inter-dependencies between the two layers of encryption before assuming one will provide a proper backup for the other. It is very important as well to try to quantify the probability of failure of a single layer of encryption as well as two layers of encryption. The difference between the two failure rates must then be compared to the extra cost of adding, configuring, and maintaining the second layer of encryption.

As mentioned earlier in Chapter IV, the two layers of encryption are not independent. Fault Tree Analysis helps to point this out. The extent to which the layers are not independent remains to be seen. We still need to fill the tree with more information. The following section will discuss some of the work that can be done to help procure some of the missing information.

B. FUTURE WORK

Much work must still be done before anything conclusive might be said about the second layer of encryption being an effective backup layer of protection. Work in Formal Methods would be useful to prove important security properties about the system such as the protocols used. Research needs to continue in cryptography. Metrics must be further explored and studied to develop new and effective metrics. Fault Tree Analysis should be analyzed more and much more work can be done to the fault tree provided in this thesis.

1. Formal Methods

One of the prominent topics discussed earlier in the thesis was the idea of dependence within a system. An analyst can find dependencies by looking for recurring or similar nodes within a fault tree. These nodes probably indicate some subsystem upon which more than one component relies. This method will also help reveal important or troublesome subsystems because the fault tree may appear more dense in that subtree than other areas.

Once one or more important areas have been identified, it may be wise to focus stronger analytical methods upon those areas. These methods may include some form of Formal Methods. If applied correctly, Formal Methods is probably the most reliable way of assuring certain properties about a system. It is also one of the most costly methods and must be used wisely.

Some of the more significant nodes of commonality are protocols, algorithms, or other form of requirements or specifications. Many things may depend upon these protocols and algorithms. As such, it may be wise to ensure that the protocols and algorithms achieve what is intended and only what is intended. If the properties of the protocols and algorithms are proved to be correct or exist, the probability of an attack on these specifications should be quite low if not zero.

Formal Methods or some other kind of systematic analysis should be applied to any protocols and algorithms that are used. Some example protocols and algorithms are provided for this thesis. Encryption and tunneling protocols include SSH, IPsec in tunnel

and transport mode, TLS, and L2TP. Key or token handling protocols include IKEv2 and Kerberos.

Research has already been started on some of these protocols. This research may help provide more insight or more areas or topics of research. Some security properties have been formally defined and proven for specific uses of IPsec (Guttman, Herzog, & Thayer, 2000). This research is somewhat out-dated and may need to be updated for the new versions of the protocol. Catherine Meadows has done some protocol analysis on IKE (Meadows, 1999). Again, this research may need to be updated for IKEv2.

We must not forget that these encryption layers are not fully independent. They, at the very least, ride on top of each other within the packet headers. So, these protocols must be analyzed together. Examples include: SSH+SSH, IPsec+IPsec, IPsec+SSH, SSH+IPsec. Before this can be done, we must know how to analyze combined protocols.

Some work has already been done to help analyze the composition of protocols. Cremers provides a brief research agenda for moving forward in this area (Cremers, 2006). Guttman and Thayer have published a paper on determining two protocols to be independent if they use encryption such that they do not overlap each other (Guttman & Thayer, 2000). Guttman has published another paper on using strand spaces for protocol composition analysis (Guttman, 2009).

When we have a better grasp of analyzing a composite of protocols and have analyzed the various combinations of the primary protocols, we can go on to add in IP, TCP, and UDP for completeness purposes.

2. Cryptography

Much of the research currently being conducted on cryptography is relevant to this topic. Researchers continue to study AES and other cryptographic algorithms to look for weaknesses and to test their strengths. This research is important to stay as up-to-date as possible. The research can provide numbers such as the estimated amount of time and effort required to brute force keys for given algorithms.

Known cryptographic attacks may be further studied and at least theoretically ap-

plied to the two layers of encryption. For example, one may study how the bit flipping weakness in CBC mode may affect the encryption schemes. A study of the information that can be revealed may provide some useful information for cracking the protocols. Similarly, one can study the information that may be revealed via the construction of bad packets and their ICMP reply messages. Many other attacks should be tested and applied to this type of architecture.

3. Metrics

As mentioned earlier, metrics and numbers are important, but it is often difficult to generate useful and meaningful ones. More research in the field of metrics may lead to a better understanding how to analyze system security strength.

There are quite a few numbers or metrics which we can generate or almost generate given the proper amount of time and access. We may analyze the different implementations of the same protocols to determine the similarities between the implementations. The survey of software implementations could include the following: IPsec, SSH, IKEv2 and other key distribution software, and cryptographic algorithms. This information could indicate the likelihood of vulnerabilities that may exist in both implementations. We may determine the number of keys needed in the system, the software required, the amount of work necessary to implement and maintain the system, the cost of the software and labor, and the work load for the encryption devices for both one and two layers of encryption. This information will help to determine the extra cost incurred by the addition of the second layer.

4. Fault Tree Analysis

Computer security researchers can pour more work into the field of Fault Tree Analysis and the fault tree started in this thesis. Quite a bit more thought must be given to understanding the complexities of systems and the dependencies between system subcomponents. Fault trees can be useful in revealing some of these dependencies as well as helping to organize information on such dependencies. This thesis provided a very brief introduction into this area of research.

The fault tree developed in this thesis is merely a beginning. Significantly more information and detail can and should be added to the tree. It may need to be re-organized or even started over again with different root nodes to further explore the topic. More types of attacks and attack paths should be added to the tree. Particularly interesting but sometimes difficult to deal with subjects are insider threats, social engineering, and tricking or fooling the target into revealing information or allowing unauthorized access to the attacker. A lot of information is already known in these areas. They have not been added to the fault tree due to time constraints.

Information gathered from the sections above may also be incorporated into the tree. Protocols that have been analyzed for certain properties via Formal Methods can help provide information as to the plausibility of certain events. If the protocol is proved to never reach a certain state or reveal important information to unauthorized individuals, the corresponding nodes in the tree may indicate such by stating the event is highly unlikely or even impossible. Information gathered from metrics could also help immensely. Information such as costs of certain events or attacks, probabilities of events, and other similar statistics may prove useful when added to the tree.

Natural disasters and failures of system components would also be an interesting area of topic. This thesis focused on failures originating for one or more attacks. For a full analysis of the viability of the second layer of encryption as a backup layer, natural disasters and failures must be analyzed. As discussed in earlier chapters, natural failure rates are more difficult to predict in computer systems than in somewhat more simple electronic or mechanical systems. Thus, a study of natural failure rates may prove interesting.

C. ARE THE BENEFITS WORTH THE ADDITIONAL COST?

This thesis has focused primarily on quantitatively or at least systematically showing the increase in benefits from one layer of encryption to the use of two layers of encryption. It is important to understand what we gain from something. It is even more important to be able to compare the increased benefits with the cost of the new component. With-

out the comparison between the two, we have no way of determining if the new layer of encryption is worth it.

This new layer of encryption comes with a number of additional costs here and there. The new software or software licenses cost money. The amount of additional labor to install, configure, and maintain the second layer of encryption costs money. The hardware used to do the encryption and decryption will have an increased workload and may need to be upgraded because it is no longer sufficient. The amount of network traffic increases due to the added packet header information, the extra keys that must be distributed, and the new sessions that need to be created. The complexity of the system also increases, often times more than we realize. Also, the new software and possibly new hardware introduce new avenues of attack and vulnerabilities to the system. As can be clearly seen, the new layer of encryption does not come without its costs.

Unfortunately, costs are very difficult to predict. Charette cites many examples of information technology projects that failed or were significantly over budget. The cost of these projects were not properly identified due to a number of reasons including the following: unrealistic or unarticulated project goals, inaccurate estimates of needed resources, poorly defined system requirements, poor communication, and the complexity of the project. He goes on to state that from 2000 to 2005 failed software projects have "cost the U.S. economy at least $25 billion and maybe as much as $75 billion". This estimate does not include the projects that exceeded their budgets or their timelines which most projects do (Charette, 2005).

D. A DIFFERENT LOOK AT BACKUP PROTECTION

Security measures must be carefully analyzed before being applied to a system. In this section, we provide another small example of a defense-in-depth mechanism. We go on to analyze it briefly talking about its strengths and weaknesses.

Let us assume some host receives some input. We want to verify that this input is correct before letting the host receive it. Adding one device, Checker 1, to check the input

before arriving at the host will help to increase the security. However, this device may be compromised or make mistakes allowing in appropriate or in accurate information to reach the host.

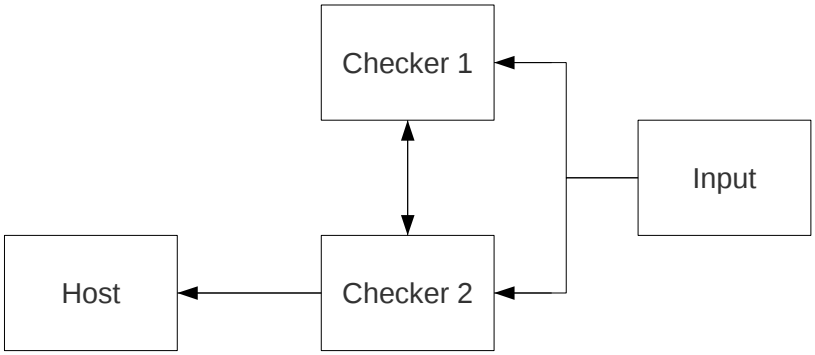

Figure 21: Another example of defense-in-depth.

To help protect the added device or cover where it fails, we can try to add a secondary, but similar device called Checker 2. Thus, both security devices receive the same input. They process the data and individually make a decision as to the validity of the data. Checker 1 will notify Checker 2 of its decision. If they both agree the data is valid, Checker 2 will send the data on to the host. This method will help to catch mistakes that might be made by a single checker.

While this method may increase some of the security of the system, it does reduce the security in other areas as well. The addition of Checker 2 (and Checker 1) introduces more avenues of attack into the system. The possibility of vulnerabilities in the system is increased. The secondary checker also increases the possibility of DoS attacks. Checker 2 must wait for Checker 1, compare the results, and make a decision thus increasing the amount of work and time required for each piece of data received. This makes it easier to overwhelm the system. Also, if one of the checkers were to be compromised, it would be simple to skew the results and never let input reach the host.

As mentioned earlier, this is an example to help reiterate concepts already discussed in the chapter. A security feature added to any system may increase the security of that system. This security feature, and any other feature, increases the possibilities of attacks on

the system as well as the complexity of the system. The benefits of the security must be carefully measured and weighed against the costs of designing, implementing, and maintaining the security feature. Any security mechanism must be carefully analyzed even ones that appear on the surface to be useful as in some defense-in-depth architectures.

E. RECOMMENDATIONS

This section provides a few recommendations. Due to the complex nature of systems, each system is different and has different requirements. Some will benefit from following one of the below recommendations. Some will not benefit or may even be harmed by following one of the recommendations. Each system's requirements must be tailored to that particular system.

One of the most important principles in computer security is simplicity. Bruce Schneier states that "security's worst enemy is complexity" (Ferguson & Schneier, 2003). Simple systems tend to make it easier easier to analyze, identify problems, and fix problems. Complex systems make it significantly more difficult to analyze, understand, and find and fix problems. The problems in a complex system may tend to be more complex and more difficult to fix than problems presented by a simple system.

This thesis was unable to quantitatively show a significant improvement in security by adding a second layer of encryption. Adding the second layer of encryption would increase the cost of the system as well as the complexity of the system. As such, this thesis recommends that the design of the system should be kept as simple as possible.

For most systems, one layer of encryption should be sufficient. The easiest to implement is the architecture with one layer of encryption at the security gateways. Of the architectures mentioned in earlier chapters, this architecture requires the fewest machines to be configured, the fewest keys to be generated and distributed, and the least amount of time and money to be consumed installing and maintaining the system. It is also the most simple architecture to understand and conceptualize. If the gateway provides encapsulation along with the encryption, the host header information may also be protected from some

traffic analysis attacks.

As mentioned in earlier chapters, a single layer of encryption at the gateway level does have some disadvantages. Traffic within the network is not protected from others listening on the network. Also, if the gateway becomes compromised, all network traffic processed by the gateway may become compromised.

If data must be protected along its entire path of travel or the data is sensitive enough, one layer of encryption at the host level may be considered. This architecture requires more work because every host must be properly installed, configured, and maintained. The concept and the design is more complicated than the single layer at the gateway level but is significantly less complex than the other architectures mentioned in this thesis. So, the system stays somewhat simple but gains a few security benefits. If one host becomes compromised, not all network traffic will become compromised. The data is now protected from the beginning to the final destination. Host header information cannot be easily protected in this architecture.

Some systems sometimes require two layers of encryption. In this case, it may be best to provide a constant and stable layer of encryption at the gateway level. The second layer of encryption may be provided on an individual basis when needed. This second layer could be an optional email encryption, secure sites, or some other readily available encryption mechanism that does not need to be implemented and maintained system wide. If the second layer does not need to be consistently monitored and taken care of, the two-layer system may be less complicated than the two-layer encryption schemes discussed in prior chapters. This hybrid architecture provides the necessary security but attempts to reduce the complexity of the system. This may be a highly situational encryption scheme and may not work for many systems. User error or training may be a issue with this system design.

APPENDIX: EXAMPLE VULNERABILITIES

CVE-2001-0376
Summary: SonicWALL Tele2 and SOHO firewalls with 6.0.0.0 firmware using IPSEC with IKE pre-shared keys do not allow for the use of full 128 byte IKE pre-shared keys, which is the intended design of the IKE pre-shared key, and only support 48 byte keys. This allows a remote attacker to brute force attack the pre-shared keys with significantly less resources than if the full 128 byte IKE pre-shared keys were used.
Published: 06/18/2001
CVSS Severity: 7.5 (HIGH)

CVE-2001-1141
Summary: The Pseudo-Random Number Generator (PRNG) in SSLeay and OpenSSL before 0.9.6b allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers.
Published: 07/10/2001
CVSS Severity: 5.0 (MEDIUM)

CVE-2002-0414
Summary: KAME-derived implementations of IPsec on NetBSD 1.5.2, FreeBSD 4.5, and other operating systems, does not properly consult the Security Policy Database (SPD), which could cause a Security Gateway (SG) that does not use Encapsulating Security Payload (ESP) to forward forged IPv4 packets.
Published: 08/12/2002
CVSS Severity: 7.5 (HIGH)

CVE-2002-0666
Summary: IPSEC implementations including (1) FreeS/WAN and (2) KAME do not properly calculate the length of authentication data, which allows remote attackers to cause a denial of service (kernel panic) via spoofed, short Encapsulating Security Payload (ESP) packets, which result in integer signedness errors.
Published: 11/04/2002

CVE-2004-0155
Summary: The KAME IKE Daemon Racoon, when authenticating a peer during Phase 1, validates the X.509 certificate but does not verify the RSA signature authentication, which allows remote attackers to establish unauthorized IP connections or conduct man-in-the-middle attacks using a valid, trusted X.509 certificate.
Published: 06/01/2004
CVSS Severity: 7.5 (HIGH)

CVE-2004-0607
Summary: The eay_check_x509cert function in KAME Racoon successfully verifies certificates even when OpenSSL validation fails, which could allow remote attackers to bypass authentication.
Published: 12/06/2004
CVSS Severity: 10.0 (HIGH)

CVE-2005-0039
Summary: Certain configurations of IPsec, when using Encapsulating Security Payload (ESP) in tunnel mode, integrity protection at a higher layer, or Authentication Header (AH), allow remote attackers to decrypt IPSec communications by modifying the outer packet in ways that cause plaintext data from the inner packet to be returned in ICMP messages, as demonstrated using bit-flipping attacks and (1) Destination Address Rewriting, (2) a modified header length that causes portions of the packet to be interpreted as IP Options, or (3) a modified protocol field and source address.
Published: 05/10/2005
CVSS Severity: 6.4 (MEDIUM)

CVE-2005-1797
Summary: The design of Advanced Encryption Standard (AES), aka Rijndael, allows remote attackers to recover AES keys via timing attacks on S-box lookups, which are difficult to perform in constant time in AES implementations.
Published: 05/26/2005
CVSS Severity: 5.1 (MEDIUM)

CVE-2007-3108
Summary: The BN_from_montgomery function in crypto/bn/bn_mont.c in OpenSSL 0.9.8e and earlier does not properly perform Montgomery multiplication, which might allow local users to conduct a side-channel attack and retrieve RSA private keys.
Published: 08/08/2007
CVSS Severity: 1.2 (LOW)

CVE-2008-0166
Summary: OpenSSL 0.9.8c-1 up to versions before 0.9.8g-9 on Debian-based operating systems uses a random number generator that generates predictable numbers, which makes it easier for remote attackers to conduct brute force guessing attacks against cryptographic keys.
Published: 05/13/2008
CVSS Severity: 7.8 (HIGH)

CVE-2008-1198
Summary: The default IPSec ifup script in Red Hat Enterprise Linux 3 through 5 configures racoon to use aggressive IKE mode instead of main IKE mode, which makes it easier for

remote attackers to conduct brute force attacks by sniffing an unencrypted preshared key (PSK) hash.
Published: 03/06/2008

CVE-2008-5161
Summary: Error handling in the SSH protocol in (1) SSH Tectia Client and Server and Connector 4.0 through 4.4.11, 5.0 through 5.2.4, and 5.3 through 5.3.8; Client and Server and ConnectSecure 6.0 through 6.0.4; Server for Linux on IBM System z 6.0.4; Server for IBM z/OS 5.5.1 and earlier, 6.0.0, and 6.0.1; and Client 4.0-J through 4.3.3-J and 4.0-K through 4.3.10-K; and (2) OpenSSH 4.7p1 and possibly other versions, when using a block cipher algorithm in Cipher Block Chaining (CBC) mode, makes it easier for remote attackers to recover certain plaintext data from an arbitrary block of ciphertext in an SSH session via unknown vectors.
Published: 11/19/2008
CVSS Severity: 2.6 (LOW)

CVE-2008-5230
Summary: The Temporal Key Integrity Protocol (TKIP) implementation in unspecified Cisco products and other vendors' products, as used in WPA and WPA2 on Wi-Fi networks, has insufficient countermeasures against certain crafted and replayed packets, which makes it easier for remote attackers to decrypt packets from an access point (AP) to a client and spoof packets from an AP to a client, and conduct ARP poisoning attacks or other attacks, as demonstrated by tkiptun-ng.
Published: 11/25/2008
CVSS Severity: 6.8 (MEDIUM)

CVE-2009-2408
Summary: Mozilla Network Security Services (NSS) before 3.12.3, Firefox before 3.0.13, Thunderbird before 2.0.0.23, and SeaMonkey before 1.1.18 do not properly handle a '\0' character in a domain name in the subject's Common Name (CN) field of an X.509 certificate, which allows man-in-the-middle attackers to spoof arbitrary SSL servers via a crafted certificate issued by a legitimate Certification Authority. NOTE: this was originally reported for Firefox before 3.5.
Published: 07/30/2009

CVE-2009-2417
Summary: lib/ssluse.c in cURL and libcurl 7.4 through 7.19.5, when OpenSSL is used, does not properly handle a '\0' character in a domain name in the subject's Common Name (CN) field of an X.509 certificate, which allows man-in-the-middle attackers to spoof arbitrary SSL servers via a crafted certificate issued by a legitimate Certification Authority, a related issue to CVE-2009-2408.
Published: 08/14/2009

CVE-2009-2474
Summary: neon before 0.28.6, when OpenSSL or GnuTLS is used, does not properly handle a '\0' character in a domain name in the subject's Common Name (CN) field of an X.509 certificate, which allows man-in-the-middle attackers to spoof arbitrary SSL servers via a crafted certificate issued by a legitimate Certification Authority, a related issue to CVE-2009-2408.
Published: 08/21/2009

CVE-2009-3455
Summary: Apple Safari, possibly before 4.0.3, on Mac OS X does not properly handle a '\0' character in a domain name in the subject's Common Name (CN) field of an X.509 certificate, which allows man-in-the-middle attackers to spoof arbitrary SSL servers via a crafted certificate issued by a legitimate Certification Authority, a related issue to CVE-2009-2408.
Published: 09/29/2009

CVE-2009-3456
Summary: Google Chrome, possibly 3.0.195.21 and earlier, does not properly handle a '\0' character in a domain name in the subject's Common Name (CN) field of an X.509 certificate, which allows man-in-the-middle attackers to spoof arbitrary SSL servers via a crafted certificate issued by a legitimate Certification Authority, a related issue to CVE-2009-2408. NOTE: the provenance of this information is unknown; the details are obtained solely from third party information.
Published: 09/29/2009

CVE-2009-3555
Summary: The TLS protocol, and the SSL protocol 3.0 and possibly earlier, as used in Microsoft Internet Information Services (IIS) 7.0, mod_ssl in the Apache HTTP Server 2.2.14 and earlier, OpenSSL before 0.9.8l, GnuTLS 2.8.5 and earlier, Mozilla Network Security Services (NSS) 3.12.4 and earlier, multiple Cisco products, and other products, does not properly associate renegotiation handshakes with an existing connection, which allows man-in-the-middle attackers to insert data into HTTPS sessions, and possibly other types of sessions protected by TLS or SSL, by sending an unauthenticated request that is processed retroactively by a server in a post-renegotiation context, related to a "plaintext injection" attack, aka the "Project Mogul" issue.
Published: 11/09/2009

APPENDIX: FAULT TREE FOR TWO LAYERS OF ENCRYPTION

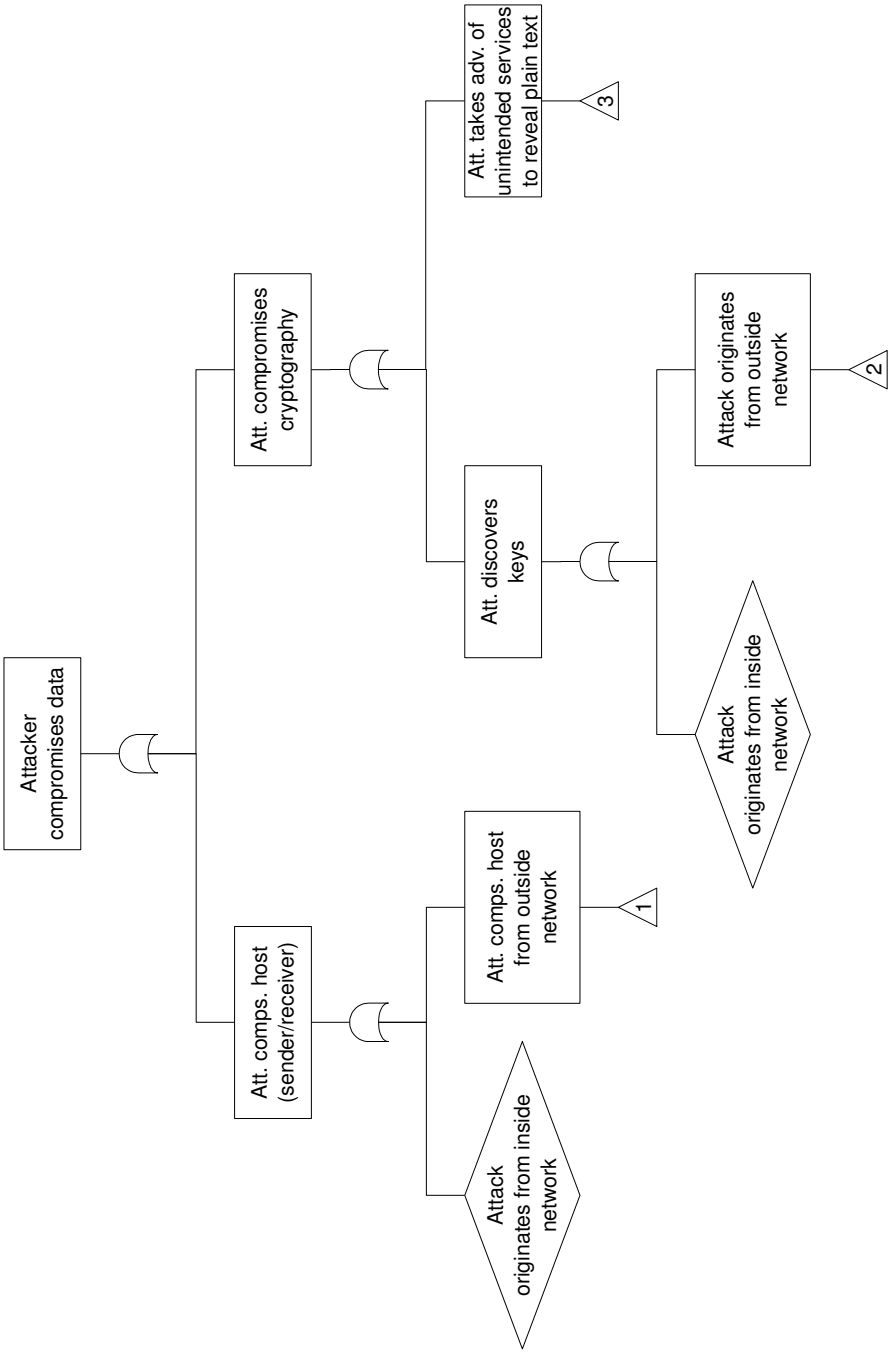

Figure 22: There are two main ways to access protected data: access it when it is not protected or compromise the protection.

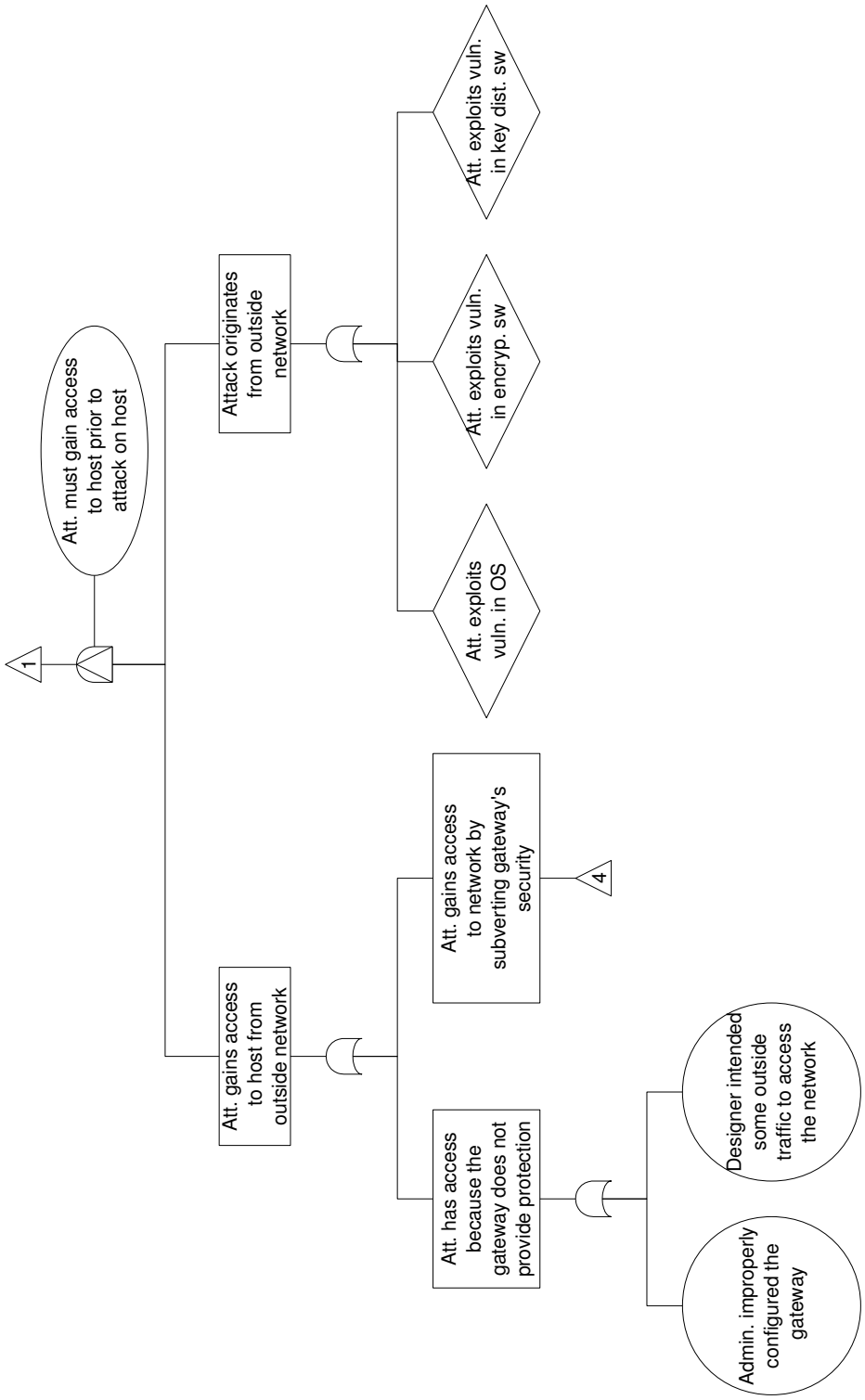

Figure 23: The attacker compromises the host machine of the sender or the receiver of the data.

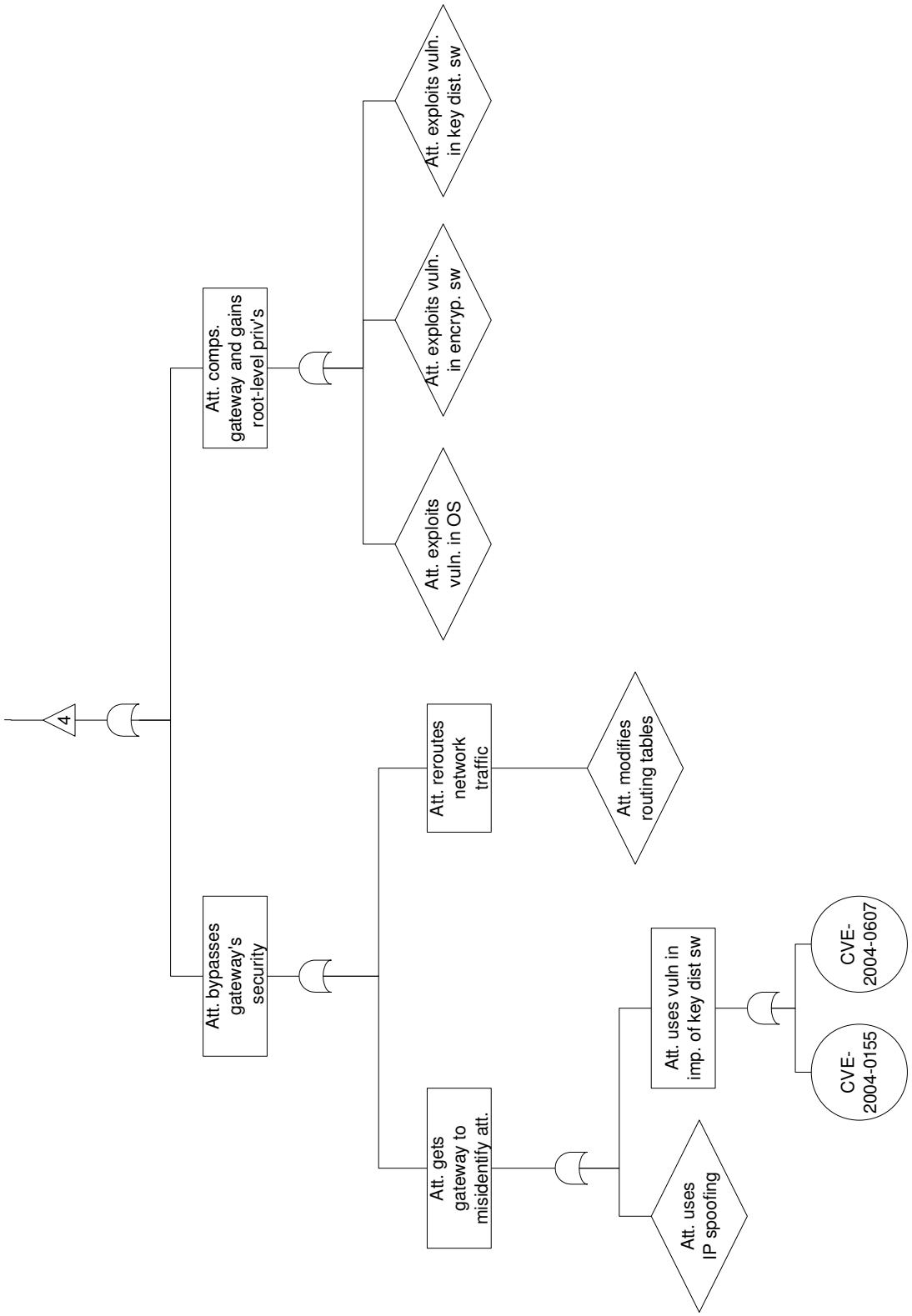

Figure 24: The attacker gains access to the host by subverting the gateway's security.

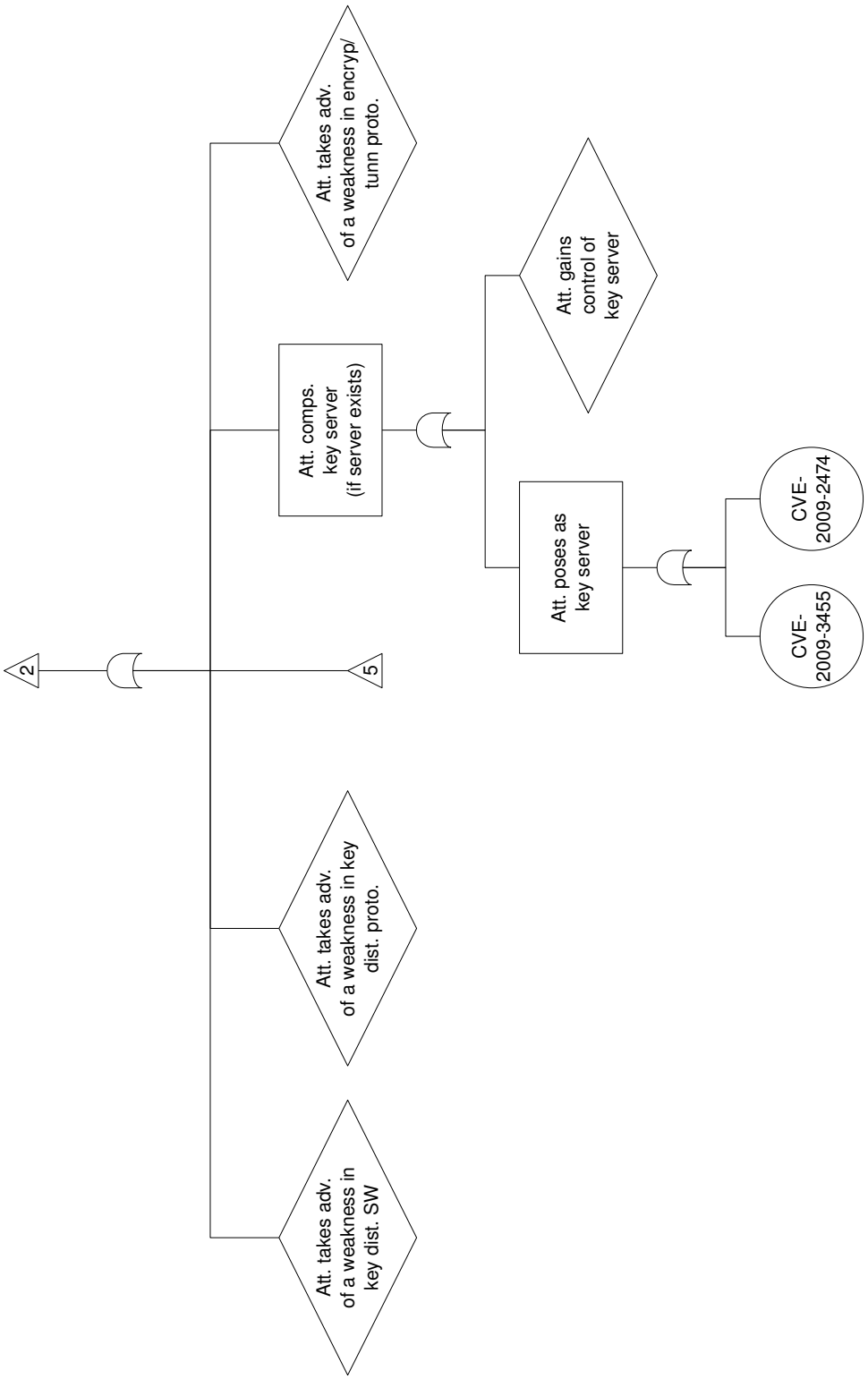

Figure 25: The attacker discovers essential cryptographic keys.

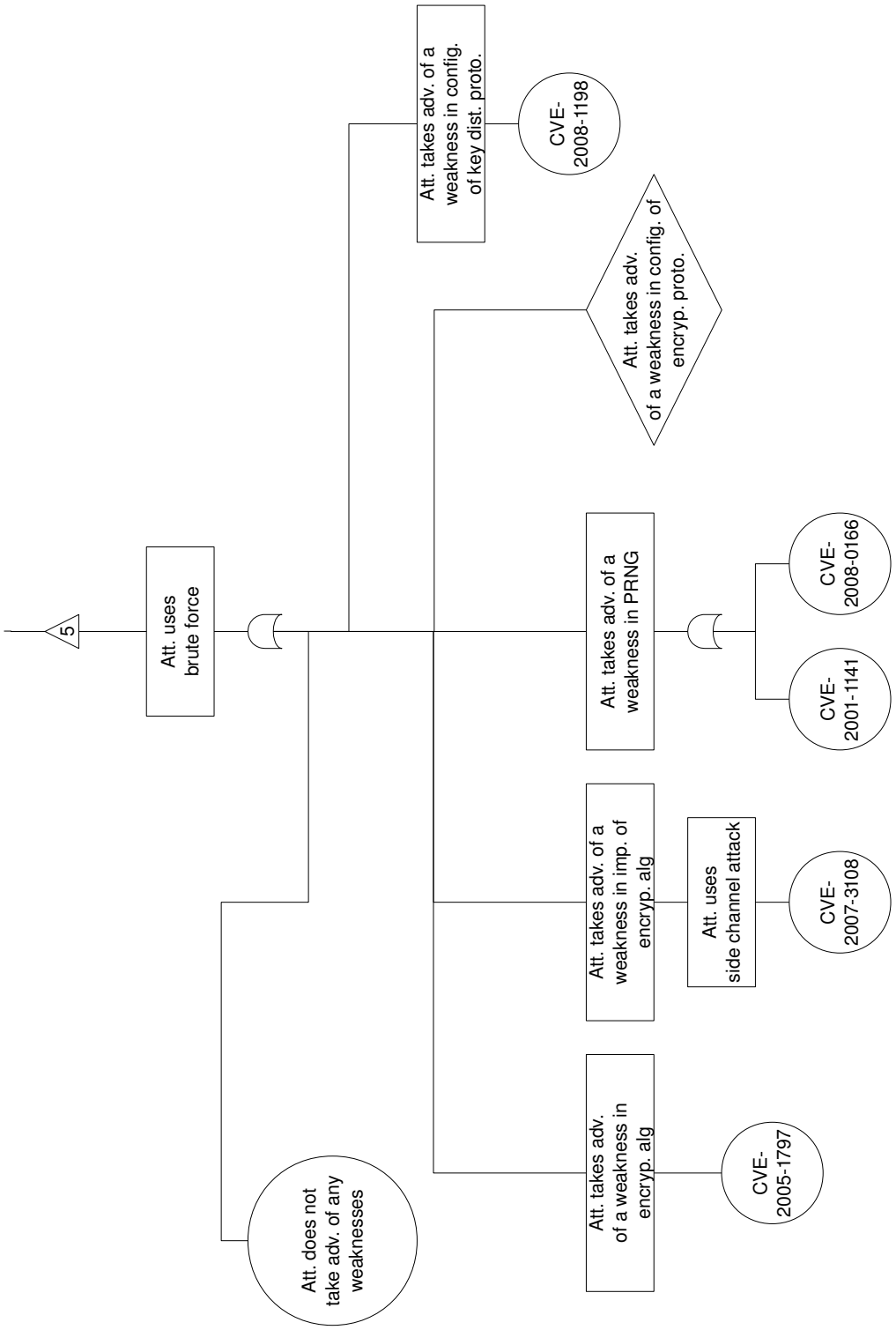

Figure 26: Attacker discovers cryptographic keys via brute force methods.

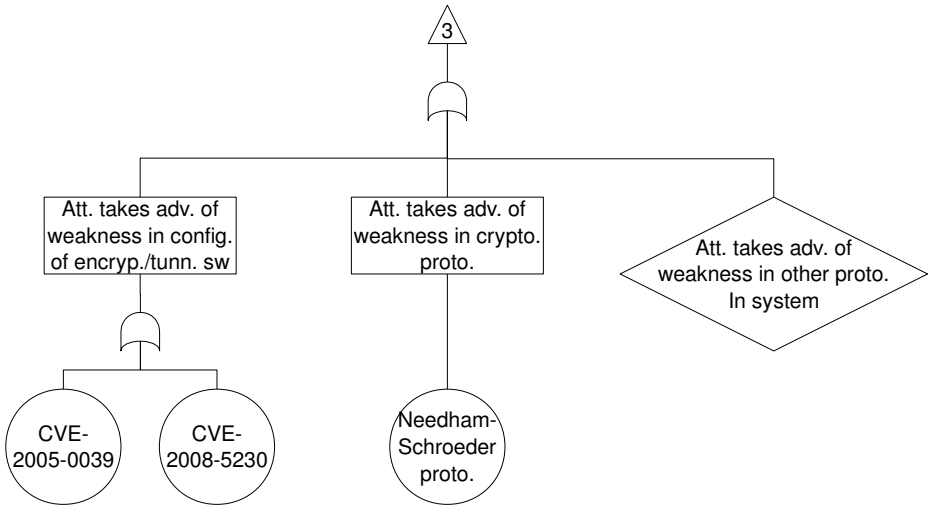

Figure 27: Attacker takes advantage of unintended services provided by the system to reveal the plaintext of encrypted messages.

LIST OF REFERENCES

Charette, R. (2005, September). Why software fails. *IEEE Spectrum*.

C. Kaufman, E. (2005, December). *Internet key exchange (ikev2) protocol* (Tech. Rep.).

CNSS. (2003). *Cnss policy no. 15, fact sheet no. 1: National policy on the use of the advanced encryption standard (aes) to protect national security systems and national security information* (Tech. Rep.). National Security Agency.

Codenomicon. (2010). *Fuzzing challenges: Metrics and coverage.* Available from `www.codenomicon.com` (Last accessed June, 2010)

Cremers, C. (2006). Compositionality of security protocols: A research agenda. *Electron. Notes Theor. Comput. Sci.*, *142*, 99–110.

Criteria, C. (2009, July). *Common criteria for information technology security evaluation.*

Denning, D. E., & Denning, P. J. (1979). Data security. *ACM Comput. Surv.*, *11*(3), 227–249.

Dierks, T., & Allen, C. (1999, January). *The tls protocol version 1.0* (Tech. Rep.).

DoDD. (2002, October). *Dodd 8500.1.*

Farinacci, D., Li, T., Hanks, S., Meyer, D., & Traina, P. (2000, March). *Generic routing encapsulation (gre)* (Tech. Rep.).

Ferguson, N., & Schneier, B. (2003). *A cryptographic evaluation of ipsec* (Tech. Rep.). Counterpane Internet Security, Inc.

Guttman, J. D. (2009). Cryptographic protocol composition via the authentication tests. In *Fossacs '09: Proceedings of the 12th international conference on foundations of software science and computational structures* (pp. 303–317). Berlin, Heidelberg: Springer-Verlag.

Guttman, J. D., Herzog, A. L., & Thayer, F. J. (2000). Authentication and confidentiality via ipsec. In *Esorics 2000: European symposium on research in computer security, number 1895 in lncs* (pp. 255–272). Springer Verlag.

Guttman, J. D., & Thayer, F. J. (2000). Protocol independence through disjoint encryption. In *In proceedings, 13th computer security foundations workshop. ieee computer* (pp. 24–34). Society Press.

Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., & Zorn, G. (1999, July). *Point-to-point tunneling protocol (pptp)* (Tech. Rep.).

Honda, O., Ohsaki, H., Imase, M., Ishizuka, M., & Murayama, J. (2005, October). Understanding TCP over TCP: effects of TCP tunneling on end-to-end throughput and latency. In *Society of photo-optical instrumentation engineers (spie) conference series* (Vol. 6011, pp. 138–146).

Kent, S., & Seo, K. (2005, December). *Security architecture for the internet protocol* (Tech. Rep.).

Landesman, M. (2010, June). *Yet another adobe zero day exploit.* Available from `http://antivirus.about.com/b/2010/06/07/yet-another-adobe-zero-day-exploit.htm` (Last accessed June, 2010)

Lowe, G. (1996). Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Tacas '96: Proceedings of the second international workshop on tools and algorithms for construction and analysis of systems* (pp. 147–166). London, UK: Springer-Verlag.

McHugh, J., Williams, J., & Skroch, M. (2000). *Information assurance metrics: Prophecy, process, or pipedream?* (Tech. Rep.). NISSC.

Meadows, C. (1999). *Analysis of the internet key exchange protocol using the nrl protocol analyzer.*

Merkle, R. C., & Hellman, M. E. (1981). On the security of multiple encryption. *Commun. ACM*, *24*(7), 465–467.

MITRE. (2002). *Proceedings: Workshop on information security system scoring and ranking* (Tech. Rep.).

Needham, R. M., & Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Commun. ACM*, *21*(12), 993–999.

NIST. (2004, February). *Fips pub 199 - standards for security categorization of federal information and information systems.*

Paterson, K. G., & Yau, A. K. (2006). Cryptography in theory and practice: The case of encryption in ipsec. In *Advances in cryptology: Eurocrypt 2006* (pp. 12–29). Springer.

Rosen, R. (2008, January). *Creating vpns with ipsec and ssl/tls.* Available from `http://www.linuxjournal.com/article/9916` (Last accessed June, 2010)

Rushby, J. (1995, August). *Formal methods and their role in digital systems validation for airborne systems* (Tech. Rep.).

Schneier, B. (1996). *Applied cryptography: Protocols, algorithms, and source code in c, second edition* (2nd ed.). Wiley.

Schneier, B. (1999, December). Attack trees: Modeling security threats.

Schneier, B. (2009, July). *Another new aes attack.* Available from `http://www.schneier.com/blog/archives/2009/07/another_new_aes.html` (Last accessed June, 2010)

Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., & Palter, B. (1999, August). *Layer two tunneling protocol "l2tp"* (Tech. Rep.).

Trochim, W. (2006, October). *Deduction and induction: Deductive and inductive thinking.* Available from `http://www.socialresearchmethods.net/kb/dedind.php` (Last accessed June, 2010)

Vesely, W., Goldberg, F., Roberts, N., & Haasl, D. (1981, January). *Fault tree handbook.*

Wing, J. M. (2007). *Scenario graphs applied to network security.*

Ylonen, T., & Lonvick, C. (2006, January). *The secure shell (ssh) protocol architecture* (Tech. Rep.).

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. George Dinolt
   Naval Postgraduate School
   Monterey, California

4. Jennifer Guild
   SPAWAR Atlantic
   Charleston, SC

5. John Mildner
   SPAWAR Atlantic
   Charleston, SC

6. James Guild
   SPAWAR Atlantic
   Charleston, SC